Allocation de ressources et ordonnancement multi-utilisateurs : une approche basée sur l'équité.

Emmanuel Medernach

N ° d'ordre : D.U. 2123
EDSPIC : 520

Université Blaise Pascal - Clermont-Ferrand II

Ecole Doctorale
Sciences Pour l'Ingénieur de Clermont-Ferrand

Thèse

présentée par

Emmanuel MEDERNACH

pour obtenir le grade de

Docteur d'Université

Spécialité : Informatique

Allocation de ressources et ordonnancement multi-utilisateurs : une approche basée sur l'équité.

Soutenue publiquement le 6 mai 2011 devant le jury composé de :

Présidente :

Claire	HANEN	Professeur des Universités, LIP6 Paris 6

Rapporteurs :

Christian	ARTIGUES	Chargé de Recherche, HDR, LAAS Toulouse
Patrick	MARTINEAU	Professeur des Universités, LI Tours

Directeurs de thèse :

Vincent	BRETON	Directeur de Recherche CNRS, HDR, LPC, Université Blaise Pascal, Clermont-Ferrand
Philippe	LACOMME	Maître de Conférences, HDR, LIMOS, Université Blaise Pascal, Clermont-Ferrand
Eric	SANLAVILLE	Professeur des Universités, LITIS, Université du Havre, Le Havre

Table des matières

Agis d'après des règles ou des maximes telles que tu puisses vouloir qu'elles soient érigées en lois générales pour toi et pour tous les autres hommes.

Ne traite jamais les êtres raisonnables, toi même, ou les autres, comme de simples moyens pour des fins arbitraires, mais comme des fins en soi.

Immanuel Kant, Principes Métaphysiques de la Morale

Résumé

Les grilles de calcul et le “cloud computing” permettent de distribuer un ensemble de ressources informatiques, telles que du stockage ou du temps de calcul, à un ensemble d'utilisateurs en fonction de leurs demandes en donnant l'illusion de ressources infinies. Cependant, lorsque l'ensemble de ces ressources est insuffisant pour satisfaire les exigences des utilisateurs, des conflits d'intérêts surgissent. Ainsi, un libre accès à des ressources limitées peut entraîner une utilisation inefficace qui pénalise l'ensemble des participants. Dans de tels environnements, il devient nécessaire d'établir des procédures d'arbitrage afin de résoudre ces conflits en garantissant une distribution équitable aux différents utilisateurs. Nous présentons une nouvelle classe de problèmes : celle des ordonnancements multi-utilisateurs. Cette thèse aborde la notion d'équité au travers de problèmes d'allocation de ressources sous incertitudes et d'ordonnancement de tâches périodiques.

Mots clefs : Allocation de ressources, Ordonnancement, Equité, Optimisation multi-critères

Summary

Grid and Cloud computing make possible the sharing of computer system resources, such as storage or computation time, among a set of users, according to their requests, thereby creating an illusion of infinite resources. However, as soon as those resources are insufficient to meet users's expectations, conflicts of interest arise. Therefore, unlimited access to limited resources may lead to inefficient usage which penalizes the whole set of users. In such environments, arbitration becomes necessary in order to settle those conflicts and ensure a fair allocation to all users. We present two classes of problems : multi-user resource allocation under uncertainty and multi-user periodic task scheduling. We tackle these problems from the point of view of fairness.

Keywords : Resource allocation, Scheduling, Fairness, Multiple-Criteria Analysis

Remerciements

En premier lieu, je suis reconnaissant envers Eric Sanlaville et Philippe Lacomme. Je remercie Eric d'avoir été présent dés le début de ce travail et de m'avoir accompagné malgré la distance. C'était un privilège de pouvoir collaborer ensemble, merci pour tes compétences, ton soutien et de m'avoir permis de pouvoir mettre en œuvre "cette idée". Merci Philippe pour avoir repris le flambeau, merci pour tes qualités de "coaching" et pour avoir su me montrer un autre point de vue pour parfaire ce travail. Merci à vous deux pour votre patience, vos suggestions et vos conseils.

Je remercie Vincent Breton pour son soutien. Merci pour son appui dans mes différentes démarches professionnelles et pour le déroulement de ma thèse.

Je remercie Pierre Reichstadt de m'avoir permis de réaliser ce travail.

Je remercie Mme. Claire Hanen, M. Christian Artigues et M. Patrick Martineau d'avoir bien voulu participer à mon jury de thèse, ainsi que pour leurs remarques constructives.

Je remercie particulièrement mon épouse pour son soutien, sa patience et son aide dans la relecture de cette thèse. Je remercie mes amis avec qui j'ai partagé différentes problématiques, merci Nicolas pour tes relectures et ton soutien, merci Marie-Jeanne pour ton exemple et ton aide, merci Hoby et Siméon : c'est à votre tour maintenant.

Un grand merci à mes sœurs pour leur compréhension et leur attention.

Introduction

Avec la création et le développement des grilles de calcul, est apparue la difficulté de l'exploitation de cette ressource. L'article de [Hardin, 1968] illustre particulièrement bien les conflits qui surgissent dès lors que des ressources doivent être partagées entre plusieurs acteurs :

> « Ruin is the destination toward which all men rush, each pursuing his own best interest in a society that believes in the freedom of the commons. »

Le libre accès à des ressources limitées entraîne inévitablement une utilisation inefficace qui pénalise l'ensemble des participants. En effet, chaque individu cherche à maximiser son intérêt personnel et aura tendance à s'approprier de manière excessive les ressources au détriment des autres [Ostrom, 2008].

Une des solution pour faire face à ce problème consiste à organiser la gestion de ces ressources. Cette gestion doit proposer des mécanismes de régulation afin de garantir une **part équitable** aux différents utilisateurs. Ainsi, cette thèse aborde la notion d'équité au travers de problèmes d'allocation et d'ordonnancement de ressources.

En effet, lors de la conception de systèmes de partage de ressources à des utilisateurs, les considérations sur l'équité font parties des préoccupations majeures. Nous avons donc analysé la situation où, un ensemble de ressources sont distribuées à un ensemble d'utilisateurs ayant les mêmes droits d'accès et, où l'équité envers les utilisateurs est un objectif souhaitable.

En raison du manque de ressources, les multiples utilisateurs sont potentiellement en conflit. Ainsi, répartir *équitablement* ces ressources constitue un problème intéressant. C'est particulièrement le cas du partage de ressources informatiques telles que le stockage ou le temps de calcul.

Dans ce travail, nous proposerons un cadre pour l'optimisation multi-utilisateurs et nous présenterons une axiomatique de l'équité. Nous appliquerons ces principes à la recherche de politiques d'allocation équitables dans un problème d'allocation de ressources et à un problème d'ordonnancement de tâches soumises en continu, modélisées par des arrivées périodiques.

En premier lieu, nous décrirons le contexte général. Nous nous poserons ensuite la question de comment répartir les ressources entre les utilisateurs et sur quels critères nous baser. Par la suite, nous introduirons une classe de problèmes qui est celle des ordonnancements multi-utilisateurs.

Dans le chapitre 2, nous présenterons les résultats obtenus pour l'optimisation multi-utilisateurs en lien avec l'équité, en se basant sur l'analyse multi-critères. Nous apporterons ensuite des résultats sur la complexité de problèmes d'ordonnancements multi-utilisateurs.

Dans le chapitre 3, nous poursuivrons notre recherche sur les problèmes d'allocation équitable où des ressources en nombre limité, doivent être attribuées de façon permanente à des utilisateurs en prenant en compte des incertitudes sur leurs demandes. Nous présenterons un algorithme d'énumération des solutions et à partir de celui-ci nous proposerons une extension dans le cas en ligne.

Dans le chapitre 4, nous traiterons de l'ordonnancement multi-utilisateurs de tâches périodiques. Premièrement, nous proposons un critère d'évaluation, puis deux modèles linéaires seront détaillés. Enfin, nous suggérerons une heuristique de calcul des solutions basées sur les heuristiques classiques de tournées de véhicules.

Chapitre 1

Contexte général

L'intérêt est ton dieu, le mien est l'équité.

Voltaire

C'est au 17e siècle que Pascal et Fermat discutent d'un problème de partage équitable posé par le chevalier de Méré, le "problème des partis". Nous parlerions aujourd'hui de problème de partage. Il s'agissait de faire le partage de sommes misées au cours d'une partie de dés de la façon la plus équitable possible. Que se passe-t-il si l'un des deux joueurs ne souhaite plus continuer, alors que le jeu est déjà commencé, que l'argent est déjà mis en jeu et qu'il n'appartient plus à aucun des deux ? A travers leur correspondance, Pascal et Fermat s'interrogent longuement pour savoir comment procéder à un partage équitable d'une somme déjà mise en jeu. Quel doit donc être la règle pour effectuer un partage équitable ?

Le partage équitable des sommes mises en jeu quand une partie est interrompue est un problème non résolu, à l'époque où Pascal et Fermat s'y intéressent. En effet, pour légaliser les pratiques de jeu à l'époque il fallait donner des conditions pour que ces jeux soient équitables. La légitimité se trouve donc liée à la notion d'équité. La façon de répartir équitablement face à l'incertain va plus loin que le seul aspect du jeu : c'est aussi un paradigme pour penser la répartition des sommes investies. Quelle doit être la juste répartition ? L'équité est la règle qui doit dicter cette répartition.

Le même genre de problèmes se rencontrent lorsqu'il s'agit de distribuer des ressources de la façon la plus équitable possible entre des personnes. L'équité est un important facteur d'évaluation dans tous les problèmes d'allocation, de distribution ou de partage de ressources entre utilisateurs ; ceci inclut entre autres les systèmes distribués comme les grilles de calcul. L'approche originale dans cette thèse est de s'intéresser au partage équitable dans le cas de l'ordonnancement. La plupart du temps en ordonnancement de production, on s'intéresse à mettre en place une organisation qui va permettre de maximiser le revenu, minimiser les coûts ou les délais. Or, c'est précisément la difficulté qui se pose dans le cas de la grille de calcul, car celle-ci permet la mutualisation à grande échelle de ressources informatiques distribuées dans le monde. Cet outil est utilisé par de nombreux scientifiques. Il se pose alors non seulement le problème de l'efficacité d'un tel outil mais aussi, et surtout de savoir répartir ces ressources de la façon la plus équitable possible. Ce thème de recherche s'élabore à partir d'une question très concrète, puisque le sujet prend sa source d'un constat de situation.

Prenons un exemple : les files d'attente ; elles font partie de la réalité quotidienne. Il est primordial pour les clients et pour les vendeurs que le temps d'attente soit le plus faible possible. Mais, il est essentiel aussi que l'ordre des arrivées soit respecté pour chaque client. De même, chacun accepte l'existence de zones prioritaires pour les clients disposant de peu d'articles. Par contre, malheur à celui qui double des personnes dans une file d'attente ; les personnes se plaignent, la

gêne occasionnée est terrible, etc. Il apparaît donc que le critère d'équité est tout aussi essentiel, voire plus important que le simple critère d'attente dans la file. Dans le cas des files d'attente, on a une règle du jeu *tacite* et enfreindre la règle est très mal perçu.

Il devient nécessaire de préciser ce que l'on entend par le terme d'équité. Au cours de cette thèse, nous avons cherché à caractériser le critère d'équité, afin d'être en mesure de comparer des allocations entre elles. Qu'est-ce qu'un partage équitable ? La définition de l'équité selon le Dictionnaire de la Langue Française d'Émile Littré (1863) est :

> « Disposition à faire à chacun part égale, à reconnaître impartialement le droit de chacun. »

La première propriété nécessaire pour l'équité est donc l'*impartialité* : sa caractéristique est de ne pas favoriser l'un au détriment de l'autre. Réciproquement, l'équité ne néglige pas certains utilisateurs au profit d'autres. Elle ne met pas en avant un groupe de personnes plutôt qu'un autre et évite toute forme de discrimination. Cela se traduit donc par la symétrie ou le fait d'être incapable de faire des distinctions entre les individus selon leur identité.

La deuxieme propriété est fondée sur la reconnaissance des droits de chacun. Or, le respect de chacun implique de garantir la protection contre l'injustice, l'oppression, ou toutes formes de domination. Il s'agit d'une égalité en droits, non pas en conditions, mais selon les besoins des individus et dans leur traitement.

L'équité recherche à aboutir à une situation au bout de laquelle aucun tort ne subsiste. L'équité est la volonté d'attribuer à chacun ce qui lui revient. Chacun peut prétendre à un traitement équitable et obtenir une part maximale tout en respectant ceux qui ont reçus moins que lui. C'est une notion d'équilibre, si la situation est modifiée alors quelqu'un s'en retrouve pénalisé.

Par ces propriétés l'équité respecte l'efficacité mais en même temps dépasse celle-ci. L'efficacité et l'équité sont des concepts complémentaires sans être incompatibles. Il n'y a pas de conflits entre efficacité et équité. L'efficacité ou le rendement d'une société détermine le niveau de vie alors que l'équité rend compte de la distribution de ce niveau de vie entre individus. Vouloir opposer équité et efficacité serait pernicieux ; cela suppose que pour être efficace il faudrait être injuste, ou bien, que pour être équitable il faudrait être inefficace. L'équité est la volonté de pousser tout le monde vers le haut sans leser personne.

Ce qui a motivé nos recherches est le fait de prendre conscience qu'un indicateur de performance (qu'il soit individuel ou collectif) peut cacher, voire encourager, des comportements abusifs alors que les utilisateurs s'attendent à un traitement impartial. Par exemple, lorsque chaque individu cherche à maximiser son indicateur de performance individuel, chacun aura tendance à exécuter plusieurs transferts en parallèle afin de monopoliser la bande passante, et ceci aux dépends des autres. Au niveau des indicateurs de mesure de performance collective, certaines tâches seront rejetées afin de maintenir un niveau de service de qualité moyenne acceptable, par exemple les mécanismes de contrôle d'admission dans les réseaux [Massoulié et Roberts, 2000; Guitart et al., 2010] ou pour les grilles de calcul [Sandholm et al., 2008]. Selon nous, les droits de chacun ne peuvent pas être méprisé pour satisfaire un bien général fictif, comme la moyenne, qui réduit somme toute l'ensemble des utilisateurs à une seule personne imaginaire. Dans ce sens, nous nous opposons à l'utilitarisme [Bentham, 1823].

Ainsi, un ordonnancement équitable envers les utilisateurs encourage leur participation, alors qu'un ordonnancement inéquitable provoque l'abandon de participation. De plus, des méthodes capables de justifier les choix effectués et de montrer que le traitement est équitable, permettent de régler les conflits.

Cela implique, entre autres, que la recherche des seuls intérêts personnels engendre le non

respect de l'équité. Il faut dépasser les simples conflits d'interêts. La volonté de ne pas nuire est donc incompatible avec la volonté de maximiser une mesure scalaire quelconque (individuelle ou globale).

Chaque utilisateur recherche indépendamment des méthodes afin de maximiser *son* profit. La dimension collective est malheureusement sacrifiée par ces raisonnements dans lesquels l'autre n'est réduit qu'à un rôle d'adversaire ou à un moyen de parvenir à ses fins, de façon interessé. En ce qui concerne les gestionnaires de sites, il en est de même s'ils cherchent à maximiser un profit certains utilisateurs risquent d'être injustement pénalisés.

Ce système encourage même des pratiques peu respecteuses des interêts d'autrui en donnant l'impression à ceux qui les exercent d'être récompensés de leurs actes. Ainsi, sans un minimum de règles justes, les droits de chacun ne sont pas protégés. Là où il n'y a pas d'équité, il n'y a pas de liberté non plus.

L'approche équitable consiste au contraire à considérer globalement l'ensemble des gains ou des pertes. Elle cherche à apporter des solutions jugées satisfaisantes pour l'ensemble selon des **règles d'équilibre et d'arbitrage** établies à cet effet. L'équité est "ce qui reconnait les droits de chacun en bonne justice". L'objectif principal de l'équité est ainsi d'établir un système de règles afin d'éviter tout préjudice contre certains utilisateurs. L'équité est différente de la loi en ce que la loi agit en aval pour faire justice après un litige, alors que l'équité agit en amont afin de résoudre un conflit. En effet, il n'y a de problèmes d'équité seulement là où il y a conflits. Afin de résoudre ceux-ci, une autorité de régulation est en général nécessaire. Un médiateur (ou gestionnaire) est alors désigné pour maintenir l'ordre, et régler les conflits. Il devra faire preuve d'équité, d'impartialité et avoir le souci d'effectuer une bonne répartition. Par ailleurs une répartition équitable ne correspond pas à l'égalité au sens strict. C'est une "juste mesure", *un équilibre*, qui permet de rendre acceptable une forme d'inégalité, lorsque l'égalité ne serait pas acceptable. Son rôle est de veiller à ce que les plus forts "n'étouffent" pas les plus faibles et à ce que les moins privilégiés ne s'approprient pas à tort la part des mieux-lotis. Cette mesure doit permettre de trouver des moyens pour subvenir aux nécessités.

L'équité exerce aussi un rôle d'arbitrage. Dans ce cas, l'arbitre a le devoir d'être impartial et donc désintéressé, et devra agir équitablement en excluant toute injustice et tout favoritisme. Mais comment une instance régulatrice pourrait-elle ensuite imposer une solution basée sur la recherche de l'équité ? La solution passe par une mise en place d'un cadre équitable et qui régule la vie en société.

Il n'y a de problèmes d'équité que lorsque les demandes excèdent les ressources disponibles. Dans ce cas de figure, la mise en place d'une *politique d'allocation des ressources* est nécessaire pour départager les utilisateurs. Ces politiques d'allocations de ressources doivent obéir à certaines règles ; ce qui permet de les automatiser, de les évaluer et de les vérifier. Le choix d'une politique pertinente relève donc d'une décision de société, d'un *choix social* [Mann, 1969; Larson, 1987; Chevaleyre et al., 2007; Fleurbaey, 2008; Ramezani et Endriss, 2010; Elkind et Lang, 2011].

Lorsque les ressources ne sont pas suffisantes pour l'ensemble des utilisateurs et qu'un conflit survient, la politique d'allocation fait alors office de *médiateur* [Schwiegelshohn et Yahyapour, 2000] et répartit les ressources entre les utilisateurs. Il nous paraît alors important de concevoir des principes d'arbitrage, qui soient basés sur la notion d'équité envers les utilisateurs. Il est indispensable de trouver des règles d'échanges équitables qui respectent chacun.

La productivité et la rentabilité ne sont pas contraires à l'équité. La productivité est à encourager mais doit être encadrée par des limites qui favorisent les relations de savoir-vivre ensemble.

Elle cherchera une équité, qui se soucie de l'efficacité, qui ne soit pas un égalitarisme strict, mais une égalité devant des droits, qui tiennent compte des situations.

D'où toute l'importance de définir des bornes ; non pas celles qui refusent le droit de chacun à la propriété, mais qui en définissent les frontières, et qui gardent les actions des hommes dans de bonnes limites, sans lesquelles il n'y a pas de liberté.

John Maynard Keynes, reconnu comme le fondateur de la macro-économie, constate que les marchés ne s'équilibrent pas automatiquement [Keynes, 1926, 1936] ce qui justifie le recours à des politiques économiques conjoncturelles. Cela implique qu'il faut mettre en place des mécanismes et que ces mécanismes doivent être élaborés en vue de parvenir à une situation équitable.

L'équité est un concept important qui apparaît à plusieurs reprises, sous diverses formes dans différents domaines de l'informatique. La formalisation précise d'un concept vague comme celui de l'équité est souvent sujet à controverses. Il n'existe pas de consensus et il est donc difficile de le définir précisément. Il existe donc de multiples définitions, chacune soulignant un aspect que recouvre cette notion. En effet, selon nous, l'équité cache différents concepts et c'est pourquoi il est absolument nécessaire de spécifier de quel concept particulier nous allons parler. Pour préciser la définition que nous utiliserons, nous allons faire un tour d'horizon des approches existantes.

1.1 Contexte de l'étude

La grille de calcul

C'est au CERN, à Genève, que le LHC (Large Hadron Collider) est entré en fonctionnement en 2008. Cet accélérateur de particules géant avec un tunnel de 27 km de circonférence, a été conçu pour la recherche de nouvelles particules dans le but de résoudre quelques-unes des questions scientifiques complexes en physique, entre autre la nature et l'origine de la masse.

En fonctionnement, le LHC produit 40 millions de collisions par seconde, issues de la rencontre de deux faisceaux de particules circulant en sens inverse à environ 300.000 kilomètres/seconde. La quantité d'informations à recueillir, à stocker et à analyser est de 15 Pétaoctets par an. L'analyse de ces collisions nécessitera une puissance de calcul soutenue, qu'il est impossible d'avoir localement. En conséquence, la seule possibilité pour le CERN fut en 2001, de commencer l'élaboration d'un outil capable de récolter assez de puissance de calcul pour ces analyses en continu.

Le CERN ne possède pas pour l'instant le potentiel de calcul nécessaire pour ces analyses, de cent mille fois supérieur à celui dont il dispose actuellement. Pour y parvenir, les scientifiques ont fait appel aux ressources informatiques des centres de calcul répartis dans le monde entier. Ainsi est né le projet européen pionnier de grille de calcul (projet européen DataGrid) qui a permis de développer, avec 21 partenaires scientifiques et industriels, un grand nombre de logiciels. Aujourd'hui, 70 organismes et institutions de 27 pays européens ont pris le relais à travers le projet EGEE qui a interconnecté, outre le CERN, une dizaine de centres de calcul, dits névralgiques, et 70 centres secondaires répartis en Europe, au Canada et au Japon, ce qui représentent quelques 40000 ordinateurs.

Comme cela a été le cas pour le Web, l'expérience de communication en informatique partagée devrait trouver de nombreuses applications dans le secteur des sciences de la vie, comme la génomique, la chimie, la climatologie, ou l'astronomie, disciplines qui demandent, elles aussi, d'énormes moyens de calcul.

La grille a donc eu comme objectif de rassembler la puissance de calcul des laboratoires parte-

naires du CERN dans le monde pour les besoins de calcul du LHC. Aujourd'hui, l'outil est ouvert à d'autres communautés scientifiques. Un des interêts de la grille de calcul est aussi de lisser les pics de demandes en répartissant la charge sur plusieurs sites. En effet, comme l'utilisation est fluctuante, la grille permet de réduire l'impact de la saturation des sites.

Le projet Enabling Grids for E-sciencE (EGEE), qui a fait suite au projet DataGrid, vise à fournir aux chercheurs universitaires et à l'industrie, un accès unifié à des ressources informatiques réparties, indépendamment de leur localisation géographique. EGEE fournit une infrastructure de grille de production pour un grand nombre d'applications scientifiques, telles que les sciences de la terre, les hautes énergies, la bio-informatique ou encore l'astrophysique.

Un site regroupe plusieurs clusters, plusieurs systèmes de stockage et une infrastructure réseau performante entre les deux. En présence de nombreuses machines le taux de pannes n'est plus négligeable et le site doit mettre en place des procédures de surveillance pour pallier à ces problèmes et garantir une certaine qualité. Les sites sont autonomes dans la gestion de leurs clusters mais ont des obligations en ressources à fournir aux groupes selon les collaborations et des engagements en terme de qualité de service. L'installation et la maintenance des logiciels utilisés sont gérés par les groupes eux-mêmes.

La grille de calcul permet le partage de ressources de calcul et/ou de stockage. Dans un tel environnement apparaît le problème de distribuer les ressources à la fois de façon efficace et équitable. Un des principes importants est que la grille doit respecter la liberté des fournisseurs. Les sites ont la liberté d'attribuer leurs ressources comme ils le souhaitent à des groupes particuliers, en revanche une fois l'attribution décidée, on doit respecter le principe d'équité entre individus égaux en droits. Le sujet de cette thèse a donc été d'explorer ce qui est nécessaire pour construire une architecture d'ordonnancement équitable et efficace.

Le premier travail consiste à définir précisément le concept d'équité et à le quantifier. L'objectif est de répondre aux demandes des utilisateurs, des groupes d'utilisateurs et des sites. La mesure choisie pour l'équité doit nous permettre de rechercher la meilleure politique d'ordonnancement au niveau local pour le double critère d'équité et d'efficacité.

Il est ainsi apparu nécessaire de repenser le problème non pas en "Comment rassembler la puissance de calcul des différents laboratoires" mais plutôt d'établir un principe d'équité : "Comment garantir un accès raisonnable à cette puissance de calcul ?".

Ordonnancement sur clusters

Un gestionnaire de batch (ou "batch scheduler") est un outil dont la fonction est d'exécuter et de placer les tâches des utilisateurs sur les machines. Il permet aux utilisateurs de soumettre et de suivre l'évolution de leurs tâches. Citons quelques batch scheduler parmi ceux les plus utilisés : BQS (du Centre de Calcul IN2P3 de Lyon), Condor [Thain et al., 2005; Tannenbaum et al., 2001], LoadLeveler [Kannan et al., 2001], LSF [Parsons et Sevcik, 1997], MAUI [Jackson et al., 2001], OAR [Capit et al., 2005], OpenPBS [Feng et al., 2007], SLURM [Yoo et al., 2003], Sun Grid Engine [Gentzsch, 2001], ... La plupart des gestionnaires de batch se composent d'un système de batch dont le rôle est d'effectuer le placement des tâches et de fournir aux utilisateurs des outils de soumission et de contrôle de leurs tâches, et d'un ordonnanceur dont le rôle est de déterminer l'ordre de placement des tâches. Le système de batch ne décide pas du placement ; c'est le rôle de l'ordonnanceur. En pratique les algorithmes de listes [Graham et al., 1979; Schutten, 1996] sont utilisés : des priorités sont attribuées aux tâches et l'ordonnanceur exécute les tâches dans l'ordre de ces priorités dès qu'une machine est disponible.

Le système de batch utilisé sur nos clusters est TORQUE [Staples, 2006], une version open-source du système de batch PBS [Jones, 2002]. Le rôle d'un système de batch est la gestion de l'envoi des tâches depuis une machine frontale (appelée "Computing Element (CE)") sur l'ensemble des machines de calcul appelées aussi "Worker Nodes (WN)".

L'ordonnanceur MAUI [Jackson et al., 2001] est fréquemment utilisé par les sites de la grille de calcul WLCG. Cet ordonnanceur fonctionne sur le principe d'une file d'attente triée selon la priorité des tâches. L'administrateur peut spécifier des pondérations selon les groupes, les utilisateurs, le temps d'attente. Il est aussi possible d'attribuer une priorité supplémentaire à une tâche, par exemple lorsque son groupe ou son utilisateur a consommé un temps de calcul inférieur à son quota de temps pendant une période. Cette fonctionnalité, appelée "Fair Share", est bien pratique lorsque les utilisateurs ont des flots continus de tâches car elle force la convergence des parts vers la part spécifiée, elle ne convient pourtant pas pour une utilisation plus irrégulière (en pics de demandes) caractéristique de l'utilisation des grilles de calcul. En effet les tâches de longues durées peuvent monopoliser un cluster et empêcher l'écoulement des autres tâches, ce qui a tendance à décourager l'utilisation de ce cluster par des utilisateurs avec des lots de tâches plus courtes.

Mécanisme d'ordonnancement utilisé dans gLite

Le middleware de grille utilisé au dessus de la partie cluster est gLite [Laure et al., 2004, 2006; Andreetto et al., 2008]. Les utilisateurs soumettent leurs tâches à un serveur "Workload Management System (WMS)" dont le rôle est de répartir les tâches sur les sites.

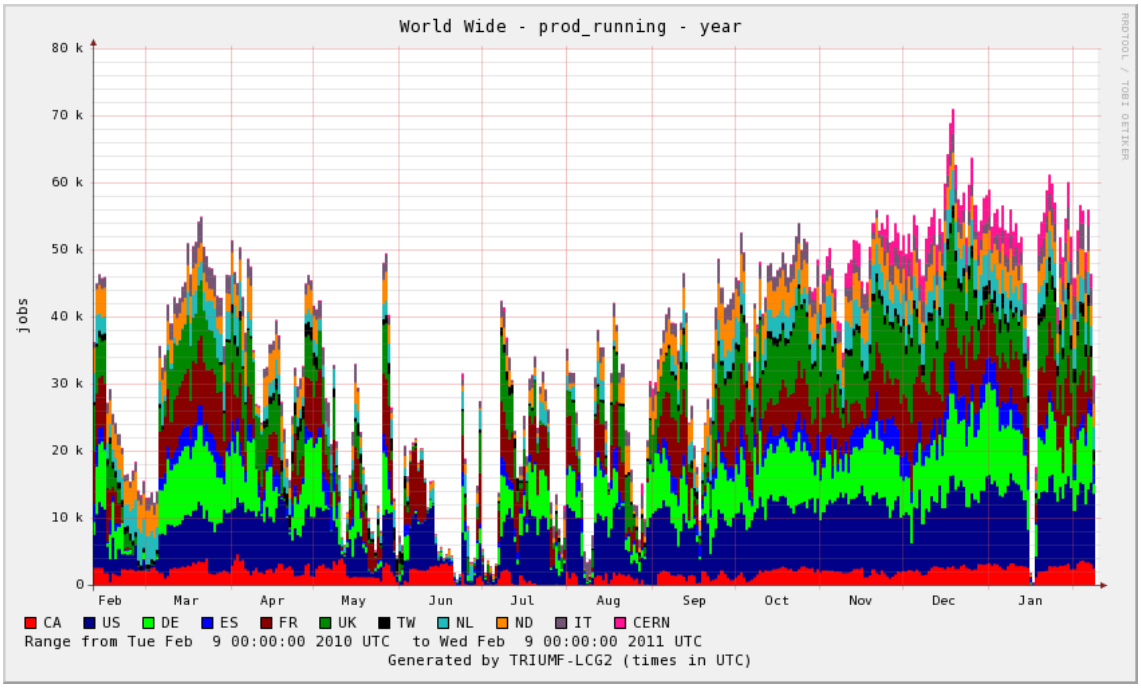
Un site est passif devant l'envoi des tâches et ne participe pas directement à la sélection par un WMS : les sites publient de l'information sur l'état de leurs clusters et le WMS gère la sélection des sites pour chacune des tâches. L'utilisateur peut spécifier certaines contraintes à respecter pour la sélection du site dans son JDL [Pacini, 2005].

Ce mécanisme de séléction présente quelques défauts : lorsque la soumission est trop rapide et que le système d'information est trop lent à présenter des données à jour, il arrive qu'un site soit submergé par des tâches ("Burst" [Cavalieri et al., 2003; Lingrand et al., 2009]) parce qu'il se trouve en tête de liste sur un WMS. Pour pallier à cela, un processus de choix aléatoire a été mis en place. Il existe un autre défaut qui vient du fait que plusieurs WMS cohabitent et ne se coordonnent pas entre eux. Bien sûr, il est aussi possible pour un utilisateur de spécifier le site de destination au lieu de laisser le WMS choisir.

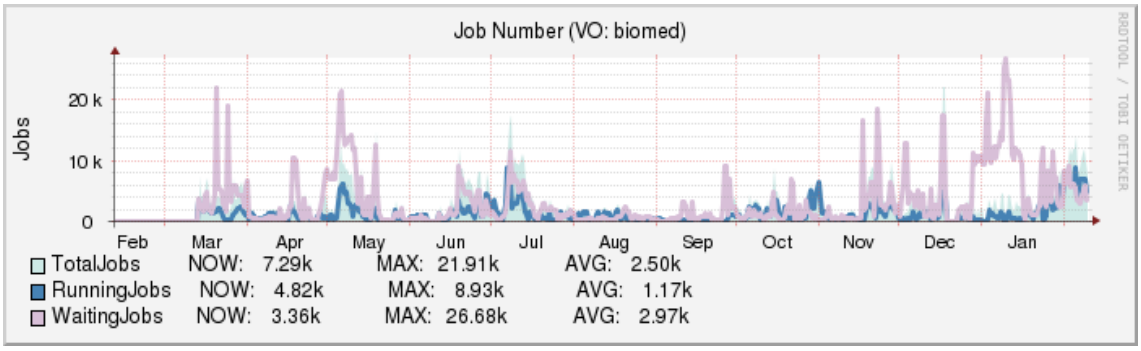
Certains groupes utilisent leur propre outil [Maeno, 2008; Moscicki et al., 2009; Vanderster et al., 2010; Tsaregorodtsev et al., 2004; Paterson et Tsaregorodtsev, 2008; Jacq et al., 2006; Jacq et al., 2007] de soumission de tâches et de gestion des priorités entre les tâches des utilisateurs d'un même groupe, selon des rôles attribués et définis par le groupe. Ces outils régulent ainsi l'envoi des tâches selon la politique d'attribution des ressources de chaque groupe à ses utilisateurs.

Diversité des tâches et des groupes

Les tâches varient selon les types d'applications (Analyse, Simulation, Reconstruction, etc.) et aussi selon leur utilisation. Par exemple, les opérations d'analyse ont des arrivées chaotiques (non régulières) et sont de durée relativement courte. De plus, les utilisateurs s'attendent à une utilisation interactive, il faut donc fournir un temps de réponse acceptable.



(a) Le groupe ATLAS



(b) Le groupe BIOMED

Fig. 1.1.1 Exemples de l'utilisation de la grille par les groupes (VO) *ATLAS* et *BIOMED*. Comme on peut le constater sur ces graphes, la grille de calcul est utilisée de façon intensive. Le premier groupe soumet des tâches de façon continue, tandis que l'utilisation de la grille par le second groupe se caractérise par des périodes d'intense activité.

Groupe	Framework	Nombre de tâches en 2010	Moyenne par jour
ALICE	ALIEN	$\approx 20 \times 10^6$	55×10^3
ATLAS	PANDA	$\approx 200 \times 10^6$	550×10^3
BIOMED	WISDOM	$\approx 2.5 \times 10^6$	7×10^3
LHCb	DIRAC	$\approx 7.5 \times 10^6$	20×10^3

Tab. 1.1 Soumission de tâches sur l'ensemble des sites

Caractéristiques des tâches

Les utilisateurs soumettent des rafales de tâches [Medernach, 2005; Li et Wolters, 2007] : un ensemble de tâches soumises à court intervalle suivi d'un temps plus long. On parlera de *paquets* de tâches d'un utilisateur. Les comportements varient selon les groupes ; certains groupes comme les groupes de Physique des Hautes Energies (HEP) envoient des quantités irrégulières de tâches en continu. D'autres groupes ont des usages de la grille en pics de demandes.

Expérimentalement, on remarque que les durées sont réparties selon une loi log-linéaire par morceaux qui dépend du groupe [Medernach, 2005; Christodoulopoulos et al., 2008] ; les durées peuvent appartenir à un ensemble d'intervalles disjoint, chacun associé avec une probabilité $\{(]t^i_{\min}, t^i_{\max}], p^i)\}$. Dans l'intervalle $]t^i_{\min}, t^i_{\max}]$ l'opération a une probabilité $p(t)$ d'être inférieure à une durée t

$$p(t) = P[\text{Durée} \leq t] = \frac{log(t/t^i_{\min})}{log(t^i_{\max}/t^i_{\min})}$$

Soit

$$t = t^i_{\min} \frac{t^i_{\max}}{t^i_{\min}}^{p(t)}$$

Cette loi signifie que dans un intervalle $[t^i_{\min}, t^i_{\max}]$ la quantité de calcul (quantité de tâches multipliée par leur durée) engendrée par les opérations de durée θ est constante : par exemple pour chaque opération de durée θ on aura en moyenne a opérations de durée θ/a.

En pratique, un site ne connait pas les durées des tâches reçues, cependant ces durées obéissent à une loi de répartition qui dépend du groupe de l'utilisateur. Les durées habituelles des tâches pour certains groupes peuvent aller jusqu'à 3 jours ou plus, d'autres groupes ont des durées de tâches autour d'une demi-heure ; il y a donc une grande diversité selon les groupes. Un mécanisme de temps limite supprime les tâches de trop longue durée en fonction du groupe d'appartenance (au delà de 5 jours par exemple).

1.2 État de l'art sur l'équité

> Il n'y a pas d'idée à laquelle l'univers songe moins qu'à celle de la justice. Il ne s'occupe que d'équilibre, et ce que nous appelons justice n'est qu'une transformation humaine des lois de l'équilibre.
>
> "La sagesse et la destinée", Maurice Maeterlinck

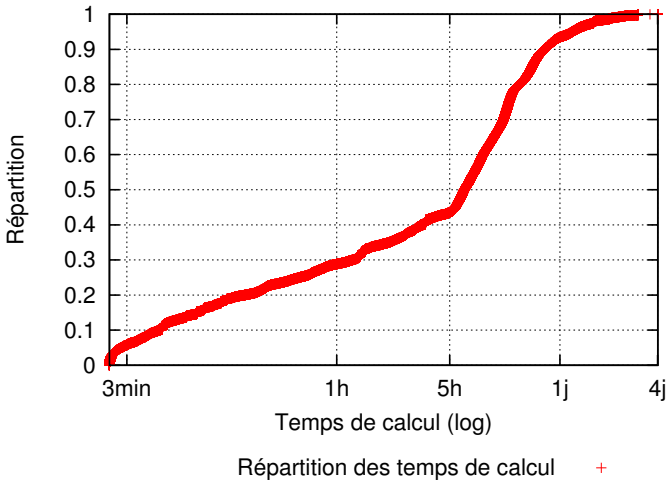


Fig. 1.1.2 Répartition des durées des tâches pour le groupe *ATLAS*

Il est particulièrement difficile de quantifier la notion d'équité car diverses définitions coexistent [Marsh et Schilling, 1994]. En effet, la notion d'équité est floue et recouvre plusieurs concepts. Les différentes définitions s'efforcent de quantifier certains aspects de l'équité. La difficulté est de séléctionner la mesure appropriée parmi la panoplie des mesures existantes pour l'équité.

Nous allons tenter de dépeindre le paysage existant et préciser ces notions avant de se focaliser sur la notion d'équité que nous avons choisi d'étudier dans cette thèse.

Il est tout d'abord nécessaire de définir envers qui doit s'appliquer l'équité. En effet, être équitable envers les tâches ou l'être envers les utilisateurs n'a pas le même sens. D'autre part, il faut identifier une mesure sur les ressources. Prenons un exemple : répartir un ensemble de marchandises dans différents entrepôts ; si l'on cherche à être équitable en volume ce n'est pas équivalent à être équitable en quantité ou en poids.

En règle générale, l'équité est vue comme une propriété vraie ou fausse ; ici elle sera considérée comme une "mesure" de la répartition des ressources envers les utilisateurs. Nous allons présenter brièvement les études existantes. Le lecteur pourra se réferer à [Lan et al., 2010] pour une liste de différentes mesures couramment employées pour l'équité.

Equité en sociologie

Des études sur l'équité ont été menées en sociologie. Pour John Ralws, philosophe américain du début du XXe Siècle, l'équité est bien plus qu'une valeur ; c'est aussi la recherche d'un cadre qui régule le conflits d'intérêts. Rawls [Rawls, 1985] met l'accent sur l'idée que la justice doit être basée sur l'équité [Rawls, 1971]. Il montre alors, que les principes d'égalité et de liberté (absence d'oppression ou de discrimination) sont des conséquences de ce concept. Il propose une conception de règles d'organisation fondamentales sur l'équité se basant sur le rejet de l'exclusion et sur le respect de la dignité humaine. Selon lui, une société juste est une société qui assure une liberté égale à ses membres. A partir de principes énoncés sur la notion de justice, il établit une hiérarchie dans les priorités : on ne saurait selon lui sacrifier un principe de plus grande priorité pour en satisfaire

un autre ; par exemple le principe d'égalité ne doit pas aller au delà du principe de liberté :

> « To respect persons is to recognize that they possess an inviolability, founded on justice that even the welfare of society as a whole cannot override. It is to affirm that the loss of freedom for some is not made right by a greater welfare enjoyed by others. It does not allow that the sacrifices imposed on a few are outweighed by the larger sum of advantages enjoyed by many. Therefore in a just society the liberties of equal citizenship are taken as settled ; the rights secured by justice are not subject to political bargaining or to the calculus of social interests. The only thing that permits us to acquiesce in an erroneous theory is the lack of a better one ; analogously, an injustice is tolerable only when it is necessary to avoid an even greater injustice. Being first virtues of human activities, truth and justice are uncompromising. »
>
> John Rawls [Rawls, 1971]

Files d'attente

[Rafaeli et al., 2002] considère l'expérience psychologique liée à l'attente, du point de vue des comportements des utilisateurs. Il montre qu'une file d'attente *prévisible* est perçue comme plus juste par les utilisateurs et qu'une file commune est considérée plus équitable que plusieurs files parallèles. Ainsi dans ce cas, on peut considérer l'équité comme une propriété de *régularité*. L'article montre un résultat essentiel qui est que le fait de gagner du temps n'est pas globalement le critère le plus important pour les utilisateurs. En effet, lorsque les utilisateurs ont le choix, ils préfèrent adopter un système plus équitable plutôt qu'un autre pouvant conduire à des injustices.

Vouloir être équitable est par conséquent rationnel : les participants à un système s'attendent tout d'abord à un traitement qui soit équitable entre eux et ensuite, à ce que la performance de ce système soit acceptable.

Ainsi de nombreux travaux [Raz et al., 2004; Sabin et al., 2004; Wierman, 2007] sur l'étude des files d'attente tentent de définir et de comparer le comportement de différentes politiques avec une politique de réference supposée idéale, par exemple "Processor Sharing".

En plus de la notion d'efficacité, un système peut mettre en place des moyens afin d'améliorer l'expérience de l'attente elle-même. [Larson, 1987] analyse les réactions psychologiques derrière les files d'attente : plus que le phénomène d'attente lui-même, l'article analyse quelles sont les attentes des clients dans une file. Il montre quelques principes simples comme l'annonce du temps d'attente ou une organisation de l'environnement d'attente pour améliorer la perception de l'attente elle-même. De même, [Mann, 1969] étudie les méthodes de régulation mises en place dans les systèmes de files d'attente en vue de régler les conflits, en diminuant la pénibilité de l'attente. Ces mécanismes ne peuvent toutefois pas se substituer à une volonté de servir efficacement les clients, ni justifier des traitements inéquitables.

Par ces exemples, nous voyons que les réflexions sur l'équité sont au moins aussi importantes que les préoccupations d'efficacité et non antagonistes.

L'équité dans les systèmes concurrents

L'équité joue un rôle crucial dans la sémantique et la vérification des systèmes réactifs [Park, 1980]. Des livres entiers sont consacrés à cette notion d'équité, (voir par exemple [Francez, 1986]).

Les chercheurs de ce domaine ont soigneusement élaborés une taxonomie des propriétés de l'équité. Celles-ci apparaissent dans la littérature, comme par exemple l'équité inconditionnelle, l'équité au sens faible, l'équité au sens fort [Francez, 1986; Varacca et Völzer, 2006], et ainsi de suite. Compte-tenu de la multitude de notions d'équité qui ont été proposées et étudiées, cette recherche s'avère importante.

L'équité est ici vue comme une propriété sur la garantie de progression d'un processus. Comme nous le voyons, cette notion d'équité est fondamentale dans la vérification et la validation de systèmes répartis. Néanmoins, nous n'avons pas affaire ici à l'équité au sens strict d'un problème de répartition entre individus mais plutôt dans le sens où on peut démontrer qu'un phénomène de famine entre processus est evité de façon certaine (chaque processus pourra être exécuté indéfiniment, ou chaque processus aura accès à la ressource infiniment souvent) mais sans précision de quantité de ressources.

Le choix social computationel

Le choix social est une branche de l'économie qui s'intèresse aux processus de décisions collectives, comme le vote, l'allocation de biens indivisibles [Maniquet, 2002; Lipton et al., 2004] ou le partage équitable de ressources informatiques. Le processus d'allocation [Chevaleyre et al., 2006] doit vérifier des propriétés d'équité. On recherche alors des solutions équilibrées.

Arrow [Arrow, 1970] pose les fondements de la "théorie du choix social" avec ses travaux sur le principe de décision collective. On parle de décisions collectives lorsque des décisions sont à prendre par un groupe d'agents sur un ensemble d'options. Chaque agent exprime ses préférences grâce à une fonction d'utilité ou plus simplement via un préordre sur les options. Notons que les options peuvent être composées, par exemple lorsque plusieurs choix non indépendants sont à prendre en même temps. Les procédures d'agrégation de préférences posent des problèmes complexes et peuvent conduire à des paradoxes délicats [Arrow, 1970; Guilbaud, 1952].

Le choix social computationnel [Chevaleyre et al., 2007] étudie les procédures de décisions, analyse leur complexité et établit des règles de décisions vérifiant des propriétés souhaitables, telle que le fait de rendre la manipulation du résultat par les agents plus difficile.

L'équité vue comme uniformité

Ce point de vue considère l'équité comme une volonté de niveler ou d'égaliser la part reçue par les utilisateurs. On parle aussi d'égalitarisme [Mortimore, 1968; Landesman, 1983; Frankfurt, 1987].

[Fagin et Williams, 1983; Ajtai et al., 1995] considèrent le problème du covoiturage et proposent un critère d'équité basé sur la différence entre le minimum et le maximum. [Raz et al., 2004, 2006; Avi-Itzhak et al., 2007] utilisent un modèle de réseaux de files d'attente multi-serveurs et proposent de mesurer l'équité envers chaque tâche. Le principe de RAQFM est de mesurer la déviation par rapport à une répartition idéale des ressources.

[Jain et al., 1984] propose un ensemble d'axiomes que doit respecter une mesure de l'équité (Invariance d'échelle, Métrique scalaire, Continuité). Ces axiomes conduisent à une mesure de l'équité (appelée mesure de Jain [Jain et al., 1984]) qui est équivalente à la mesure de l'angle θ entre le vecteur des utilisateurs et le vecteur unité $[1, 1, \dots]$: $\frac{(\sum x_i)^2}{n \sum x_i^2} = cos^2(\theta)$.

L'équité vue comme proportionnalité

Ce point de vue définit l'équité comme la nécessité de répartir proportionnellement à une quantité.

Le "Proportionate-progress fair scheduling" (ou P-Fair scheduling") [Baruah et al., 1993; Aoun et al., 2008] consiste à trouver des ordonnancements périodiques avec préemption des tâches où chacune des tâches doit se terminer avant l'occurence de la tâche suivante. L'objectif est de construire des solutions qui soient les plus *régulières* possibles, dans le sens où à tout instant le rapport entre quantité de calcul traitée par tâches et le temps écoulé doit se rapprocher le plus possible du rapport entre la durée de la tâche et la période. Ces ordonnancements préemptifs sont à rapprocher de "Processor Sharing".

[Megiddo, 1977] propose un algorithme pour le calcul d'un flot sur un réseau en maximisant une métrique des débits entre les sources et les puits du réseau.

[Nieh et al., 2001; Wong et al., 2008] quant à lui, propose un ordonnanceur proportionnel pour le noyau Linux avec une complexité constante en temps.

Équilibrage de charge entre utilisateurs : L'équité comme critère d'optimisation

Une des caractéristiques critique d'un ordonnanceur est qu'il soit équitable [Kay et Lauder, 1988; Lauder, 1995]. [Kay et Lauder, 1988] remarquent que les ordonnanceurs sont traditionnellement conçus pour affecter les ressources de façon équitable aux processus, au regard de quoi ils proposent "SHARE" un ordonnanceur qui tient compte des utilisateurs et des groupes. Il est devenu apparent que la notion d'équité doit s'appliquer de préférence aux utilisateurs et non aux processus [Larmouth, 1974, 1978; Newbury, 1982; Woodside, 1986]. En effet un ordonnanceur équitable envers les processus a tendance à favoriser les utilisateurs qui lancent beaucoup de processus en allouant le CPU proportionnellement au nombre de processus ce qui risque d'étouffer les utilisateurs avec peu de tâches.

[Casanova, 2006] utilise le ralentissement moyen ("average stretch") sur l'ensemble des tâches pour évaluer les performances d'un ordonnancement. L'équité est mesurée par le coefficient de variation. Ainsi l'auteur de cet article se place du point de vue de l'école *égalitaire* de l'équité.

Par exemple, le point de vue égalitariste cherche à ce que la part de chaque utilisateur converge vers la part qu'il a payée : si un utilisateur a payé trois fois plus qu'un autre sa part doit converger vers trois fois la part de l'autre utilisateur. Ce point de vue, tout à fait acceptable, pose plusieurs problèmes : tout d'abord, lorsque le premier utilisateur ne consomme pas la part auquel il a droit. Il est alors difficile de justifier qu'on cherche à ce que le rapport des parts soit proche de 3 car cela risque de pénaliser l'autre utilisateur en l'empêchant d'utiliser des ressources libres. D'autre part, dans ce genre de système à proportionnalité les utilisateurs qui ont des moyens importants peuvent étouffer les autres utilisateurs et monopoliser les ressources. Nous proposons plutôt de chercher à équilibrer une fonction de comparaison pondérée avec la part de chaque utilisateur.

Pour pallier à ce problème certains ordonnanceurs utilisent la notion de déterioration [Jackson et al., 2001], celle-ci consiste à pondérer les parts des utilisateurs selon leur historique. Cette modification du mécanisme d'ordonnancement montre plutôt une défaillance de la notion d'égalitarisme lorsqu'elle est appliquée de cette façon. La notion d'équité a pour objectif de mieux préciser ces concepts.

La notion d'équité dans les problèmes d'optimisation dans les télécommunications.

TCP assure le contrôle de la congestion dans le transport des paquets réseaux. Le modèle impose de dégrader équitablement le service lors d'une congestion, plutôt que de refuser l'accès à des flots supplémentaires ("admission control"). Le principe se base sur l'équité Max-Min [Bertsekas et Gallager, 1987]. L'équité Max-Min [Bertsekas et Gallager, 1987; Nace et al., 2006] s'applique lorsque l'objectif est de distribuer équitablement un ensemble de ressources partagées par des agents. [Bertsekas et Gallager, 1987] donnent une solution centralisée à ce problème.

[Kelly, 1997; Kelly et al., 1998] propose de mesurer l'équité dans le problème de partage de la bande passante grâce à une fonction d'utilité logarithmique ; ce qui correspond au concept d'équité vu comme proportionnalité ("Proportionnal Fairness"). [Massoulie et Roberts, 1999] étudie ce problème avec d'autres fonctions d'utilité comme la moyenne harmonique ; ce qui correspond à minimiser la somme des délais.

[Kumar et Kleinberg, 2000] ont défini la notion de "global approximation" pour l'équité en travaillant sur des vecteurs par utilisateur. La notion provient du concept de "max-min fairness" utilisé lors de la conception de réseaux. Cependant, cette méthode est trop sommaire, dans le sens où elle approche coordonnées par coordonnées tous les vecteurs de l'ensemble de solution et cherche celui avec le ratio le plus petit. L'optimum est donc dépendant de la forme de l'ensemble des solutions et peut être médiocre en comparaison du maximum pour l'équité. De fait, cela revient dans notre cas à chercher un vecteur proportionnel aux demandes.

[Goel et al., 2001b] ont proposé un objectif plus fort, qui est de calculer une solution qui soit une approximation par coordonnée de **tous** les vecteurs d'allocation possibles. Avec cette mesure, le vecteur max-min peut être de mauvaise qualité aussi.

[Kostreva et Ogryczak, 1999; Kostreva et al., 2004; Ogryczak et al., 2008] traitent de problèmes d'allocation équitable de ressources, comme le partage de bande passante, où les critères sont comparables. Leur approche se base sur l'optimalité de Pareto et le principe de transfert de Pigou-Dalton pour aboutir à l'ordre de Majorization. Ils définissent un critère d'agrégation de type OWA, qui est un compromis entre l'équité et la moyenne.

Equilibrage de charge

Dans le cas de l'allocation équitable d'une quantité de travail à du personnel, ou dans le cas où on cherche à répartir équitablement une quantité de calcul sur plusieurs machines, on cherche à minimiser la charge maximale, ce qui correspond au problème d'équilibrage de charge.

Le problème d'équilibrage de charge en présence de tâches temporaires a été étudié dans [Azar et al., 1997; Lam et al., 2002] : des tâches préemptives arrivent en ligne (leurs caractéristiques sont connues seulement à leur arrivée et à un instant donné on ignore quelles seront les tâches futures.) L'objectif est de minimiser la charge maximale, sur les machines. Dans notre cas, nous étudierons le partage de charge entre utilisateurs plutôt qu'entre machines.

Dans [Golovin, 2005], un problème d'allocation de ressources indivisibles est traité où chaque utilisateur a une utilité additive sur les ressources et l'objectif est de maximiser l'utilité minimale.

[Dhakal et al., 2007] expose un algorithme d'équilibrage de charge coopératif dans un environnement distribué, sujet à une incertitude sur les délais dûs aux communications entre les nœuds et à une incertitude sur les durées des tâches. Il observe que lorsque les informations entre les nœuds participants ne sont pas fiables ou anciennes, les nœuds peuvent échanger inutilement de la charge

entre eux.

[Pruhs et al., 2004] analyse le ratio de compétitivité de différentes politiques d'ordonnancement en ligne et fournit des bornes selon le nombre de machines, ou si les tâches sont préemptives, etc.

[Woeginger, 1997] donne un PTAS pour le problème de la maximisation de la charge minimale. De façon similaire [Epstein et Van Stee, 2008] donne aussi un PTAS pour ce problème mais avec des agents "égoïstes".

De façon similaire, lorsqu'il est question de répartir équitablement un ensemble de machines de différentes vitesses entre utilisateurs, on cherche à maximiser la part de l'utilisateur le moins bien servi. Cela conduit à des problèmes de maximisation de la charge minimale $P_m||\max C_{\min}$ [Haouari et Jemmali, 2008]. Ce problème, aussi appelé problème de recouvrement [Tan et He, 2004], est NP-difficile et est comparable au problème du Makespan mais où les tâches sont de durées négatives. [Woeginger, 1997] donne un PTAS pour ce problème et [Skutella et Verschae, 2009] un PTAS robuste (stable en nombre de migrations de processus). [Csirik et al., 1992] montre que LPT a un ratio de compétitivité de $(3m-1)/(4m-2)$. [Lin et al., 1998] étudie le même problème avec des dates de disponibilité des machines, où il est montré que LPT a un ratio de performance de $(2m-1)/(3m-2)$.

[Azar et al., 2000] s'intéresse au problème de la répartition du placement aléatoire d'objets dans n boîtes, problème que l'on retrouve par exemple dans la création de tables de hachage efficaces. A la fin du placement, la boîte la plus remplie contiendra avec une haute probabilité $(1+o(1))log(n)/log(log(n))$ objets. Cet article montre que si, à chaque nouvel objet, deux boîtes sont tirées au hasard et si cet objet est placé dans la boîte la moins remplie ; alors cette quantité devient $log(log(n))/log(n)+O(1)$, ce résultat est encore vrai lorsque le processus est itéré indéfiniment avec des suppressions d'objets.

Théorie des jeux

Définition 1.1 (Equilibres de Nash)**.** *Un équilibre de Nash est un ensemble de stratégies tel qu'aucun joueur ne puisse améliorer sa condition en changeant sa stratégie lorsque les stratégies des autres joueurs demeurent les mêmes.*

Dans le cas où les tâches appartiennent à des utilisateurs indépendants, [Angel et al., 2009] considèrent le problème de concevoir des mécanismes d'ordonnancement qui incitent les utilisateurs à déclarer le plus exactement possible la durée de leurs tâches. Dans ce type de problèmes, il faut idéalement garantir le fait qu'un utilisateur n'a aucun interêt à abuser le système avec de fausses déclarations. Le problème est étudié aussi du point de vue des équilibres de Nash, c'est à dire que le mécanisme doit garantir que la sincérité de tous les acteurs est un équilibre de Nash (aucun utilisateur n'a interêt à mentir si les autres ne mentent pas).

Le problème de la répartition dans les jeux de coalition

Après avoir constaté que la coopération est la base de nos sociétés, [Axelrod, 1984] pose la question de savoir sous quelles conditions la coopération peut elle émerger à partir d'agents "égocentrés" et en l'absence d'une autorité d'arbitrage ?

Par exemple prenons le jeu du dilemme du prisonnier, ce problème montre que deux personnes peuvent ne pas coopérer même lorsque que c'est leur meilleur interêt, dès lors qu'il existe un risque

qu'une personne puisse trahir l'autre et que ce comportement lui soit plus avantageux que la coopération. Ceci illustre que les seuls concepts d'*équilibre* de la théorie des jeux ne conduisent pas nécessairement à des allocations qui seraient pourtant préférées par tous les joueurs. Lorsque le jeu est répété la situation est différente, [Axelrod, 1984] montre expérimentalement que la meilleure stratégie est la stratégie de "réciprocité altruiste" : coopération au premier coup puis choisir indéfiniment ce que l'autre joueur a choisi au premier coup avec éventuellement coopération de façon aléatoire. Ainsi le fait que les participants puissent se rencontrer à nouveau incite à la coopération. [Axelrod, 1984] conclut en disant que "l'essence de la coopération ne réside pas vraiment dans la confiance, mais plutôt dans la pérennité de la relation". Un autre résultat intéressant est que la confiance entre les participants n'est pas nécessaire pour aboutir à la coopération car il suffit d'un mécanisme de représailles immédiates pour rendre les tentatives de trahison inutiles.

[Rzadca et al., 2007; Rzadca, 2007] étudie du point de vue de la théorie des jeux le problème de l'ordonnancement équitable entre organisations qui coopérent, il montre que lorsque ce problème est décentralisé il est équivalent au dilemme du prisonnier, de plus l'équilibre de Nash conduit à des pertes de performance significatives. Cependant les remarques précédentes montrent que le jeu du dilemme du prisonnier répété incite à la coopération.

Jeux de négociations et arbitrage de Nash ("Nash Bargaining Solution" ou NBS)

Ce concept de solution a été utilisé pour le problème de formation de réseaux coopératifs [Avrachenkov et al., 2010] où il faut répartir des coûts de participation, il a aussi été utilisé pour les problèmes de partage de bande passante dans un réseau [Touati et al., 2001, 2006].

Un problème de négociation part d'une situation fixée $[\dots, d_i, \dots]$ et cherche à trouver parmi l'ensemble des solutions non dominées une solution acceptable pour les différentes parties.

Nash considère que l'ensemble des solutions est convexe car aléatoirement composé de stratégies mixes dont la valeur correspond à l'espérance de cette distribution de probabilités. En supposant que cela ait un sens pour un problème donné, Nash propose un axiome selon lequel l'optima trouvé ne doit pas dépendre de l'*unité de mesure* des critères de chaque utilisateur (ou utilités). C'est l'axiome d'indépendance à toute transformation affine des utilités : en l'absence d'unité de mesure objective pour chaque critère, il est naturel d'imposer une invariance par transformation affine ($[\dots, x_i, \dots] \mapsto [\dots, \alpha_i x_i + \beta_i, \dots], \forall i, \alpha_i, \beta_i > 0$). Il montre que seul la maximisation du produit des différences avec $[\dots, d_i, \dots]$ satisfait cet ensemble d'axiomes.

1.3 Examen des approches existantes

Critique de l'approche égalitariste

La mesure de Jain [Jain et al., 1984] ($\frac{(\sum x_i)^2}{n \sum x_i^2}$) a pour nous le désavantage d'être invariante lorsque le vecteur est multiplié par une constante. Par exemple les vecteurs $[1, 2, 3]$ ou $[3, 6, 9]$ sont indiscernables. On peut considérer que cette mesure est plutôt une mesure de l'*égalité*, tout comme l'est la variance ou le coefficient de variation [Sauve et al., 1980; Wong et Lang, 1982; Wong et al., 1982; Deng et al., 2009].

Equilibre résultant d'un rapport de forces			
Proportionnalité	[Aristote, -340]	$X_i = C\frac{w_i}{\sum_i w_i}$	pas toujours Pareto
Théorie des jeux	[Nash, 1951]	Equilibres de Nash	pas toujours Pareto
Utilitarisme : maximiser un critère d'aggrégation			
Utilitarisme	[Bentham, 1823; Mill, 1863]	$\max \sum_i X_i$	Pareto
Négociation	[Nash, 1950a]	$\max \prod_i (X_i - X_i^0)$	Pareto
OWA	[Yager, 1988; Roy, 2007]	$\max \sum w_i \vec{X}_i$	Pareto
Egalitarisme : rechercher de l'uniformité			
Egalitarisme	[Mortimore, 1968; Landesman, 1983]	$\max \min_i X_i$	pas toujours Pareto
Indice de Jain	[Jain et al., 1984]	$\max(\sum X_i)^2 / \sum X_i^2$	pas toujours Pareto
Files d'attente	[Avi-Itzhak et al., 2007]	min Variance(X)	pas toujours Pareto
Approches économiques (ordres sur $\mathbb{R}^N$)			
Efficacité	[Bauchet, 1965]	$\prec_c$	Pareto
Transfert	[Pigou, 1912; Dalton, 1920]	$\prec_D$	Pareto
Majorization	[Goel et al., 2001a]	$\prec_{Maj}$	Pareto
Leximin	[Sen, 1970; Rawls, 1971; Kolm, 1997]	$\prec_{Lex}$	Pareto

Tab. 1.2 Tableaux récapitulatifs des différentes approches sur l'équité.

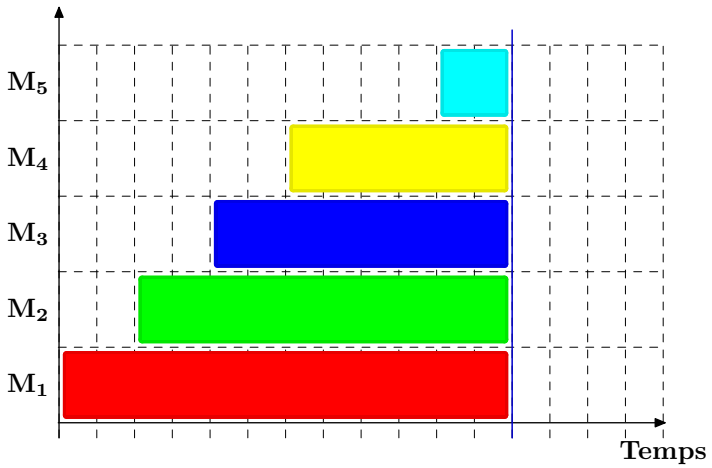


Fig. 1.3.1 La solution proposée est **égalitaire** car toutes les tâches se terminent en même temps. Pourtant elle n'est pas **équitable** car les tâches des utilisateurs auraient pu s'exécuter plus tôt. Cet exemple illustre le défaut de critères égalitaires comme le critère du coefficient de variation ou le critère de Jain.

La figure 1.3.1 montre le défaut d'une approche égalitariste de l'équité. Sur la figure, 5 utilisateurs avec une tâche chacun se partagent 5 machines. L'ordonnancement de la figure impose à chaque tâche de se terminer à la date 12, alors qu'une solution qui commence chaque tâche à la date 0 semble plus appropriée. En effet, selon nous, l'équité n'impose pas de dégrader arbitrairement les performances des utilisateurs, ce que l'égalitarisme a tendance à provoquer puisque l'ordonnancement de l'exemple est optimal pour des critères comme le critère de Jain ou ceux basés sur le

coefficient de variation.

Critique de l'approche proportionnelle

Selon nous, faire dépendre la notion d'équité des demandes des utilisateurs ne nous paraît pas justifié. En effet, si plusieurs utilisateurs ont reçu la même part, alors il n'est pas justifié que l'un d'eux proteste à cause d'une injustice uniquement parce qu'il avait demandé plus au départ : sa demande, plus importante, ne lui donne pas le droit de passer devant les autres ; ceci en accord, selon nous, avec le principe d'équité de traitement. En revanche, il nous paraît plus correct de mesurer les parts grâce à une mesure pondérée si besoin et d'équilibrer celle-ci.

[Sandel, 1984] précise d'ailleurs cette idée, dans le fait qu'une répartition qui se veut équitable ne doit pas prendre pour principe les mesures arbitraires des différents utilisateurs, mais un principe inconditionnel :

> « Differents persons typically have different desires and ends, and so any principle derived from them can only be contingent. But the moral law needs a categorical foundation, not a contingent one. »

Nous avons par conséquent recherché à ce que les parts des utilisateurs soient elles-mêmes directement inter-comparables grâce à un *principe d'évaluation* (voir la section "Optimisation multi-utilisateurs"). Ce principe d'évaluation consiste en une mesure qui, tout en tenant compte des différents interêts des utilisateurs, autorise une comparaison des satisfactions obtenues par les utilisateurs de façon impartiale. Selon nous, l'équité se définit *relativement* à ce principe d'évaluation.

Notons que la recherche d'une solution proportionellement équitable peut s'exprimer via ce principe d'évaluation en appliquant des pondérations, on recherchera alors à équilibrer un vecteur compte-tenu de coefficients. Cependant ces derniers ne devraient pas être sous l'influence des utilisateurs. En effet, si les utilisateurs peuvent ajuster eux-mêmes leurs coefficients, chacun aurait tendance à surestimer ses besoins et à négliger ceux des autres : un tel système risque par exemple de favoriser ceux qui demandent le plus ou inversement.

Critique de l'approche des équilibres de Nash

Nash [Nash, 1950b] propose la notion d'équilibre de Nash : un équilibre est atteint lorsqu'aucun participant n'a d'interêts à modifier sa stratégie. Cette notion d'équilibre est adaptée pour le cas des décisions distribuées mais n'a aucune raison d'être équitable du fait de la place accordée à l'influence des participants, comme le montre l'exemple suivant :

> « Dans les échanges internationaux les prix pratiqués ne sont pas équitables : Un pays pauvre ne peut pas jouer sur le marché des matières premières puisqu'il n'a pas de réserves. Il doit vendre sa production sans attendre. Il est, de ce fait, la proie toute trouvée de spéculateurs habiles qui joueront à la baisse. Imaginons d'ailleurs un cas d'école : deux joueurs jouant à pile ou face. Difficile de faire plus équitable ! Maintenant supposons que la mise est de 1 euro et que l'un des joueur possède une fortune de 2 euros au départ, tandis que l'autre possède une fortune de 10 euros. Naturellement si l'un des deux joueurs est ruiné la partie s'arrête. Au bout de dix coups, le joueur le moins riche sera ruiné dans plus de la moitié des cas tandis que le joueur le plus riche ne sera ruiné que dans un cas sur

> mille. Le jeu semble équitable mais le résultat ne l'est pas. En tout cas, il est certain que le joueur qui sait qu'il risque gros hésitera avant de courir un tel risque. Cela permettra au plus riche de l'intimider en le menaçant de le forcer à jouer. »
>
> "Une prise de risque inégale", Frédérick De Coninck

Par ces exemples, nous voyons que baser l'équité sur un rapport de forces (la possibilité donnée aux participants de punir un autre en changeant de stratégie) ne nous semble pas être un principe légitime puisque certains peuvent avoir potentiellement plus d'influence et l'utiliser aux dépens des autres. Comme le remarque ironiquement Pascal :

> « Ne pouvant faire que ce qui est juste fût fort, on a fait que ce qui est fort fût juste. »
>
> Pensées, Pascal

à savoir que le pouvoir ou l'influence de participants quelconques ne doit pas justifier une mauvaise répartition des ressources.

Dans le cas de jeux non-coopératifs, [Kameda et Altman, 2008] montrent un problème pour lequel l'équilibre de Nash est toujours une solution dominée. Ainsi la compétition entre participants non-coopératifs ne garantit pas d'aboutir à une solution efficace. Le problème intervient lorsque la poursuite des interêts de chacun conduit à un résultat médiocre pour tout le monde.

Ces équilibres non-coopératifs peuvent même être très inefficaces [Rzadca et al., 2007]. On définit le *prix de l'anarchie* comme étant le rapport entre la performance de l'équilibre de Nash et la performance obtenue par coopération.

Par exemple dans le cas d'ordonnancement "égoïste" ("selfish scheduling" [Czumaj et al., 2002]) le prix de l'anarchie est $\Theta(log(m)/log(log(m)))$. Le prix de l'anarchie peut même être non borné dans certains problèmes de routage [Papadimitriou et Valiant, 2009], il est intéressant de constater qu'en ajoutant un coût d'utilisation sur les routeurs alors le prix de l'anarchie devient 1, c'est à dire, comme le remarquent Papadimitriou et Valiant, que "Prices achieve the social optimum in routing". Dans le cas d'équilibrage de charge sur grille de calcul, [Ayesta et al., 2009] montre que le prix de l'anarchie est d'au moins $\frac{K}{2\sqrt{K}-1}$ pour K répartiteurs ("Resource Broker") et ne dépend pas du nombre de serveurs. Par conséquent, il est important et nécessaire que des mécanismes de coordination (idéalement décentralisés) entre répartiteurs soient mis en place sur la grille de calcul. [Beaufils et Mathieu, 2002] remet en question l'utilisation de la notion d'équilibres de Nash dans le cas des jeux répétés avec interaction entre participants.

Citons un exemple numérique simple donné par la table 1.3 qui illustre et souligne notre propos.

	A^1	A^2
B^1	$[15, 15]$	$[10, 25]$
B^2	$[2, 20]$	$[1, 10]$

Tab. 1.3 Exemple de jeux à 2 joueurs (A en bleu et B en rouge), la table indique les gains obtenus par les joueurs selon leurs couleurs respectives. Ce jeu n'a qu'un équilibre de Nash, indiqué en jaune.

Sur cet exemple nous remarquons que :
– $A^1B^2 : [2, 20]$ est le seul équilibre de Nash.

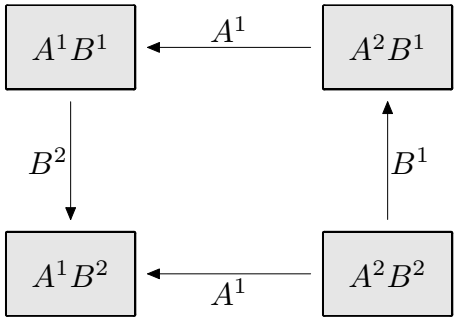


Fig. 1.3.2 Un arc indique qu'il est plus intéressant de changer de stratégie pour un des participants.

– $A^2B^1 : [10, 25]$ domine $[2, 20]$, ainsi les équilibres de Nash ne sont pas en général non dominés, i.e. il se peut qu'il existe des solutions plus efficaces (de meilleure qualité pout tout les participants).

Par conséquent cet exemple illustre le fait que les équilibres de Nash fournissent des solutions résultant d'un *rapport de forces*, conséquence du conflit d'intérêts entre les joueurs, et que ces équilibres ne constituent pas les bases d'un accord équitable dont l'objectif est au contraire de réduire les tensions. Ainsi dans cet exemple la solution $A^1B^1 : [15, 15]$ paraît être la solution la plus équitable et sera probablement celle choisie entre participants au terme d'une négotiation raisonnable. Remarquons que cela nécessite que B s'interdise de changer de stratégie [1] pour ne pas pénaliser A.

Malgré l'inefficacité des mécanismes de concurrence (caractérisés par l'absence d'une autorité de régulation) face aux mécanismes de coordination et compte-tenu des remarques précédentes, il est regrettable que certains auteurs continuent d'affirmer que l'équité envers les utilisateurs est fournie par la notion d'équilibre de Nash.

Critique de l'approche de l'arbitrage de Nash (NBS)

La proposition d'arbitrage de Nash s'applique parfaitement dans le cas de négociations où les critères des parties ne sont pas directement comparables et dont l'unité peut être modifiée par transformation affine sans pour autant changer le résultat. Pourtant on peut considérer le point produit comme un pis-aller, dans le sens où les critères (utilités) sont présupposées comme n'ayant pas d'unités fixes : c'est effectivement le seul point sur lequel les participants peuvent parvenir à trouver un accord. Ce choix particulier d'axiomes aboutit fatalement à ce résultat. D'autres choix d'axiomes ont été proposés, voir [Kalai et Smorodinsky, 1975].

Néanmoins l'axiome d'invariance est contestable car il conduit à des solutions considérées du point de vue du produit comme équivalentes mais qui ne le sont pourtant pas : par exemple s'il existe des répartitions où un des utilisateurs peut obtenir d_i et pas les autres alors toutes ces répartitions sont équivalentes au sens de Nash. De même selon l'arbitrage de Nash la répartition $[1, 100]$ est équivalente à $[10, 10]$ avec $D = [0, \dots]$ du point de vue de l'équité, ceci est difficilement acceptable car cela sous-entend qu'on puisse donner aussi peu qu'on veut à certains participants tant que d'autres ont proportionnellement plus. On peut alors s'interroger sur le caractère affiché

1. ou plus pragmatiquement de déclarer illégale la stratégie B^2 et d'appliquer des sanctions si B décide de l'appliquer.

comme rationnel des axiomes présentés. De plus, il suffit pour un utilisateur de modifier la fonction de son critère, sans en changer l'ordre, pour influer sur le résultat en sa faveur. En effet maximiser le produit n'est pas invariant lorsqu'on change par exemple l'unité de mesure par une échelle logarithmique (comme le pH pour l'acidité ou la magnitude pour des mesures d'amplitude sismique) les résultats sont alors différents. Pourtant ces transformations sont croissantes et conservent l'ordre entre les éléments. Ainsi, l'axiome d'indépendance à l'échelle [2] est trop restrictif, il nous est apparu plus essentiel de pouvoir comparer objectivement la condition des utilisateurs entre eux, plutôt que de supposer des utilités incomparables. D'autant plus qu'en pratique la procédure d'arbitrage est équitable en fonction d'un critère affiché. Nous n'avons donc pas retenu l'axiome d'invariance à l'échelle mais nous avons considéré comme plus important le fait que la solution soit indépendante de toute transformation croissante appliquée *uniformément* sur tous les utilisateurs. Cela signifie que nous nous sommes basé sur la comparaison entre utilisateurs, comparaison fondée sur un *principe d'évaluation* qui établit un ordre à partir d'une répartition donnée et qui permet de décider qui sont les utilisateurs les moins bien servis. L'axiome d'invariance d'échelle est donc discutable dans notre situation dans la mesure où nos critères sont basés sur un principe d'évaluation objectif de la part reçue par les utilisateurs et n'ont donc aucune raison de subir des transformations distinctes selon les utilisateurs. Nous imposons alors que le critère est un critère de comparaison entre utilisateurs et que la solution doit être invariante par application uniforme de toutes fonctions croissantes. Cela revient à considérer uniquement l'ordre entre les élements.

D'autre part le principe de la fonction d'utilité est d'associer à la part reçue par un utilisateur une valeur qui mesure la satisfaction de cet utilisateur pour ce gain. Cette notion fait dépendre l'allocation de la subjectivité des différents participants et ainsi parvient à un critère dont la mesure pour une même allocation dépend de l'utilisateur considéré, ce qui de notre point de vue sur l'équité est tout à fait inacceptable. De plus il est possible qu'un utilisateur puisse manipuler l'allocation obtenue en modifiant sa fonction d'utilité.

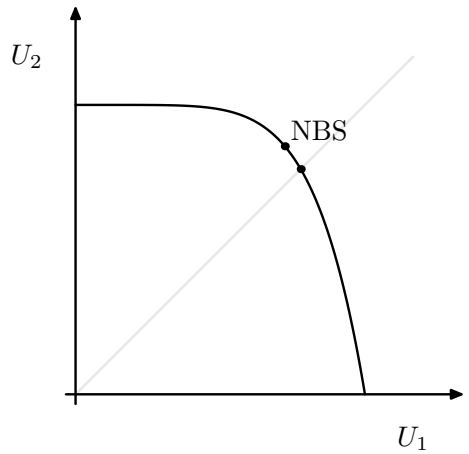


Fig. 1.3.3 Exemple d'arbitrage de Nash, le produit est maximal pour le point $[0.67, 0.80]$. La solution qui égalise les utilités est $[0.72, 0.72]$. On voit ici qu'il suffit que l'utilisateur Y modifie sa fonction d'utilité pour faire pencher la solution en sa faveur.

2. Encore une fois, ceci est la conséquence directe du fait d'avoir des *utilités* par essence incomparables car subjectives.

NBS est équivalent à $\max \prod_i (x_i - d_i) \Leftrightarrow \max \sum_i \ln(x_i - d_i)$ c'est à dire à maximiser une moyenne de logarithmes. NBS a donc les mêmes imperfections que les critères basés sur la maximisation d'une moyenne. Un exemple est donné sur la figure 1.3.3.

Ken Binmore a déclaré dans [Binmore, 1991] que

> « Certains auteurs ont mal interprété les motivations de Nash dans la formulation de sa solution d'arbitrage des jeux de négociations. Ils ont imaginé que ces axiomes pouvaient raisonnablement s'interpréter comme les critères d'un "arbitrage équitable", alors que d'autres systèmes d'axiomes ont été introduit afin de caractériser d'autres formes de négociations basées sur un arbitrage équitable. Il n'y a aucune raison qui dicte le fait que la négociation d'agents dits "rationnels" doit aboutir à une négociation équitable. »

En conclusion, nous pensons que NBS est utilisable uniquement comme terrain d'entente lorsque les critères des différents participants ne peuvent pas être comparables.

1.4 Optimisation multi-utilisateurs

Chaque tâche a en plus de ses caractéristiques habituelles la notion d'appartenir à un utilisateur particulier. Les utilisateurs se partagent les ressources offertes, le rôle du gestionnaire de ressources est de répartir ces ressources aux utilisateurs de façon à respecter l'engagement du contrat d'utilisation et d'être en mesure de justifier ses choix en cas de contestation. Le gestionnaire se doit de garantir l'égalité de droit d'accès et d'utilisation des ressources conformément à la définition de ces droits que donne la charte qui lie les participants.

Nous nous sommes intéressés à la notion d'**équité** entre utilisateurs. Cette notion est vue ici de façon générale, de telle sorte que nous considérons l'équité comme relative à un **principe d'évaluation** [Sen, 1970] permettant de décider de façon impartiale qui, entre deux utilisateurs, est le plus lésé par les choix effectués. Ce principe d'évaluation est spécifié par le contrat qui lie les utilisateurs de la ressource. Le gestionnaire du site aura donc pour mission d'être le plus équitable possible envers les utilisateurs *relativement* à ce principe d'évaluation.

Il est important lorsqu'on parle d'équité de pouvoir comparer le niveau de service donné à différents utilisateurs afin de pouvoir constater lequel est le moins bien traité. Cette comparaison peut s'effectuer via une fonction de mesure du niveau de service par utilisateur. Par principe d'objectivité, cette fonction ne doit dépendre pour un utilisateur donné que de la part reçue par cet utilisateur [Agnetis et al., 2009] et ne dépend pas des parts reçues par les autres utilisateurs. Le choix de cette fonction est primordiale afin de permettre la comparaison des situations des utilisateurs, elle classe les utilisateurs selon la qualité de service qu'ils obtiennent. L'arbitre publie le principe de comparaison choisi pour la situation donné, ainsi on peut imaginer que différents systèmes puissent avoir différents principes de comparaison, chacun basé selon l'objectif de ce système. Les utilisateurs pourront par exemple décider entre un système basé sur les temps d'attente ou un autre basé sur le temps de séjour. La différence entre un principe de comparaison et l'approche multi-critère est que le principe de comparaison sert au gestionnaire pour effectuer des comparaisons de niveau de service entre utilisateurs, de plus il est équivalent par application d'une même fonction croissante à tous les utilisateurs. Le gestionnaire établit un principe de comparaison en concertation avec les utilisateurs en fonction de leurs objectifs : il peut être une combinaison des objectifs des participants, l'important est de pouvoir établir des comparaisons objectives entre les parts reçues. L'équité se définit ensuite par rapport à ce principe de comparaison : il donne

directement un moyen de comparer directement et effectivement la qualité de service obtenue par chacun des utilisateurs. Donnons quelques exemples.

Exemples

Du point de vue utilisateur le Makespan est considéré comme négatif ($\min\max C_i^{\text{Tâche}}$), mais du point de vue des sites la quantité de travail à effectuer peut être considérée comme positive (on cherche à rentabiliser en maximisant l'occupation des machines) $\max\min C_j^{\text{Machine}}$. On voit ici que pour un même problème il existe plusieurs points de vue.

Équité en Makespan : lorsque les dates de disponibilité des tâches sont dues à des retards et que seul compte pour les utilisateurs le fait de terminer leurs tâches au plus tôt, on pourra utiliser le Makespan comme critère de comparaison.

Équité en Attente : La comparaison s'effectue sur le temps d'attente des tâches, défini comme la différence entre la date de début d'exécution et la date de disponibilité de la tâche.

Équité en Temps de séjour : on considère avec ce critère de comparaison que deux utilisateurs ont une qualité de service équivalente lorsque les temps entre l'arrivée et l'achèvement sont identiques.

Équité en Débit de calcul (inverse du "stretch" [Bender et al., 1998]) : ce critère est équivalent à une vitesse d'exécution. Dans la perspective qu'un utilisateur qui obtient un débit de calcul faible aura tendance à changer pour un site offrant un débit plus élevé.

Notons que le principe de comparaison est un ordre total entre les utilisateurs pour une répartition donnée, cet ordre n'est pas obligatoirement un ordre basé sur la comparaison d'une fonction scalaire même si en pratique c'est souvent le cas. Cette comparaison indique quel est l'utilisateur le moins bien servi et dont la condition est donc à améliorer en priorité.

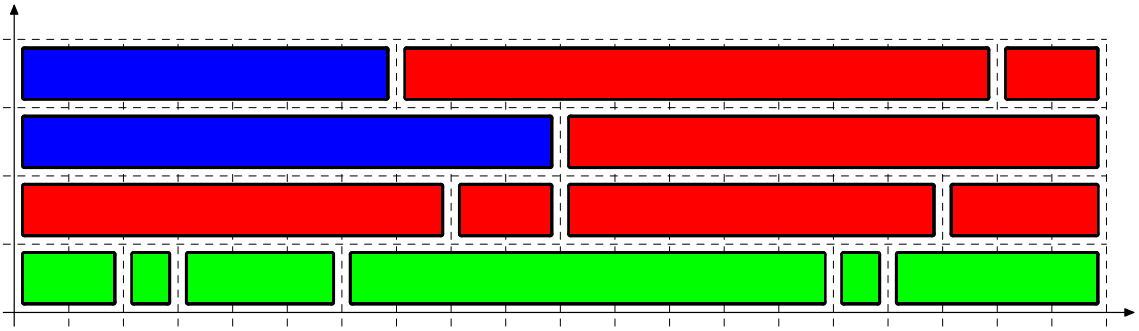
Par exemple, le principe d'évaluation peut dépendre de la quantité, du prix, du volume, etc. et pour chacun de ces principes d'évaluation nous avons un problème de répartition équitable distinct avec des solutions différentes et chacune de ces solutions est équitable selon le principe d'évaluation retenu.

Le principe d'évaluation peut ainsi dépendre de la comparaison directe de caractéristiques entre utilisateurs (i.e. comparer x_i et x_j). Il peut aussi dépendre de pondérations w affectées aux utilisateurs, lesquelles représentent la fraction relative qui revient de droit à chacun (i.e. comparer x_i/w_i et x_j/w_j).

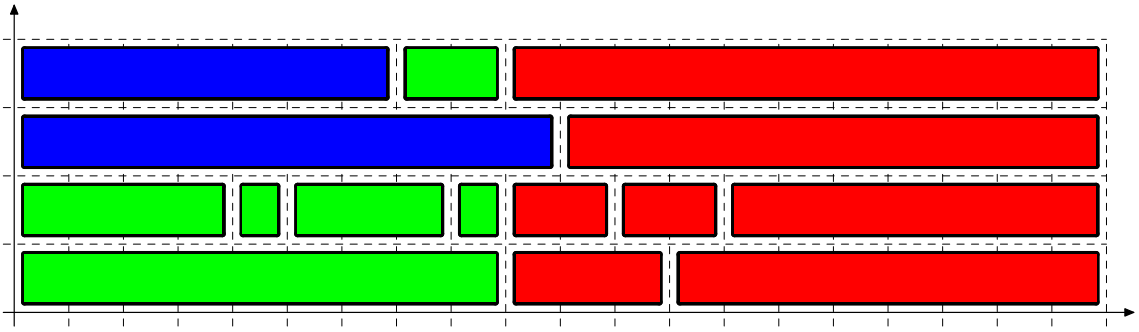
Le critère de mesure de qualité d'un ordonnancement est l'agrégation de critères par utilisateur [Blythe et al., 2003]. En vue du partage équitable des ressources entre utilisateurs nous utiliserons un principe d'évaluation du niveau de service fourni pour tous les utilisateurs ; en sorte que des comparaisons de niveau de service entre utilisateurs soient possibles. Ce principe d'évaluation peut être une agrégation de différents critères selon les besoins des utilisateurs ou des différents domaines d'applications utilisés ; l'important étant de pouvoir comparer le niveau de service obtenu par ces utilisateurs et ainsi de pouvoir décider quel est le moins bien servi. Evidemment selon le critère retenu pour cette comparaison différents choix d'ordonnancements (ou allocations) seront possibles.

Principe d'équité retenu dans cette thèse

Lorsqu'on minimise le makespan pour un problème donné, on satisfait le système car l'ordonnancement obtenu est alors le plus "compact" possible. Mais on ne satisfait pas forcément les utilisateurs dans leur ensemble car certains peuvent être servis moins bien que d'autres. Prenons l'exemple de la figure 1.1(a), le makespan est de 20 et est minimal. Par contre, l'utilisateur Vert sera mécontent car ses tâches ne s'exécutent que sur une machine et le makespan de ses tâches est de 20. Pour l'ordonnancement de la figure 1.1(b) le makespan de l'utilisateur Vert est amélioré sans changer le makespan des tâches des autres utilisateurs.



(a) Cet ordonnancement est inéquitable pour l'utilisateur Vert



(b) Amélioration de l'ordonnancement précédent

Fig. 1.4.1 Exemples d'ordonnancements avec 3 utilisateurs et 4 machines. Durées des tâches pour l'utilisateur Bleu {7, 10}, Rouge {11, 2, 10, 7, 3, 2, 8}, Vert {2, 1, 3, 9, 1, 4}.

Ainsi cet exemple illustre que minimiser une fonction économique globale (comme le Makespan) qui ne prend pas en compte la notion d'utilisateur, peut aboutir à une situation inéquitable pour certains d'entre eux. Étant donné qu'un utilisateur a la possibilité d'envoyer plusieurs tâches, une méthode d'agrégation est nécessaire pour évaluer le critère, sur chaque utilisateur, en fonction de l'exécution de ses tâches. Un ordonnancement qui souhaite garantir une équité entre utilisateurs nécessite une étude plus approfondie.

Notre approche consiste à spécifier les propriétés que requiert l'équité. Nous proposons les propriétés suivantes :

– On souhaite comparer les allocations obtenues par les utilisateurs de façon équitable, la

qualité de l'allocation pour chaque utilisateur ne doit pas dépendre de ce que les autres utilisateurs ont obtenus.
- Indépendance vis à vis du nombre d'utilisateurs : si on rajoute un utilisateur avec une allocation fixe x (peut être nulle) à deux vecteurs, cela ne doit pas modifier l'ordre d'équité entre ces vecteurs. Par exemple, l'ordre entre les vecteurs $[1,2,3,4]$ et $[5,6,7,8]$ doit être identique à l'ordre entre les vecteurs $[1,2,3,4,x]$ et $[5,6,7,8,x]$, pour tout $x \in \mathbb{R}$.
- Par souci d'impartialité, échanger deux utilisateurs doit aboutir à un vecteur équivalent. Ainsi nous rechercherons un ordre d'équité invariant par permutation des utilisateurs.

Evaluation de règles de priorité

Afin d'illustrer notre propos, nous présentons ici les performances de diverses politiques d'ordonnancement (hors ligne et en ligne) obtenues sur les logs [Medernach, 2005] d'un cluster participant à la grille de calcul EGEE/WLCG. Toutes les tâches sont des tâches mono-processeurs.

Présentons quelques politiques évaluées. Les politiques hors-ligne nécessitent la connaissance globale des informations des tâches, comme leurs durées et leurs dates de disponibilité. Par exemple lorsqu'un calendrier de prévision d'exécution est établi, les tâches sont en fait des plages horaires à placer. Dans ce modèle les caractéristiques des tâches sont connues et aucune incertitude ne subsiste, la politique hors-ligne effectue un ordonnancement des tâches et les tâches seront exécutées selon les placements prévus. On parle aussi d'ordonnancement clairvoyant [Motwani et al., 1993].

Les politiques hors-lignes que nous avons considérées sont les suivantes :

GEFT ("Global Earliest Finish Time") exécute les tâches selon l'ordre des $r_i + p_i$ croissants.

STPT ("Shortest Total Processing Time") place les tâches des utilisateurs selon l'ordre des Q_h croissants, où $Q_h = \sum_{i \in U_h} p_i$ désigne la quantité de calcul totale de l'utilisateur h. Cette politique est hors-ligne car elle place en premier toutes les tâches de l'utilisateur qui a la quantité de calcul la plus petite, puis continue sur l'utilisateur suivant, etc.

SPT-Local ("Shortest Total Processing Time Local") est une politique de liste dans laquelle on maintient une file d'attente par utilisateur : à chaque instant t où une machine se libère, cette politique exécute, parmi l'ensemble des premières tâches reçues pour chaque utilisateur et non exécutées, celle qui minimise la quantité $Q_h + \max(t, r_i)$. Chaque utilisateur voit donc ses tâches s'exécuter dans l'ordre de leur arrivée.

Lorsque le fait de connaître les informations des tâches est impossible à l'avance, une politique d'ordonnancement en ligne décide de l'exécution d'une tâche sur la prochaine machine libre parmi les tâches disponibles.

Nous avons évalué sur cet exemple les politiques en ligne suivantes :

FIFO ("First In First out") exécute les tâches dans l'ordre de leur arrivée.

STPT-LPT ("Shortest Total Processing Time Largest Processing Time") compare les utilisateurs présents selon l'ordre des Q_h croissants et lorsqu'une machine se libère exécute la tâche de plus grande durée pour l'utilisateur selectionné.

STPT-Local-Online ("Shortest Total Processing Time Local Online") Cette politique suppose que la quantité de calcul totale Q_h est connue pour chaque utilisateur, sans que les durées des tâches ne le soient. STPT-Local-Online exécute les tâches à l'instant t dans l'ordre des $Q_h + \max(t, r_i)$ croissants.

STPT-NON-CLAIRVOYANT ("Shortest Total Processing Time Non Clairvoyant") Si Q_h est inconnue au départ, soit $Q_h(t)$ la somme des durées des tâches de l'utilisateur h exécutées avant t. STPT-NON-CLAIRVOYANT exécute la tâche à l'instant t qui minimise $Q_h(t)$.

SPT-ONLINE ("Shortest Processing Time Online") exécute sur la prochaine machine libre la tâche de durée minimale parmi les tâches disponibles.

LPT-ONLINE ("Longest Processing Time Online") exécute sur la prochaine machine libre la tâche de durée maximale parmi les tâches disponibles.

STPT ("Shortest Total Processing Time") compare les utilisateurs présents selon l'ordre des Q_h croissants et exécute ensuite la tâche de plus grande durée pour l'utilisateur selectionné.

Le débit de chaque utilisateur pour des politiques hors-lignes est indiqué sur la figure 1.2(a) ; les utilisateurs sont triés par ordre croissant pour le débit. La politique STPT-LOCAL est de bonne qualité pour les 20 premiers utilisateurs cependant les 20 utilisateurs suivants ont un débit réduit de moitié par rapport au débit obtenu avec la politique GEFT. En revanche, la politique GEFT donne en comparaison pour les 20 premiers utilisateurs des débits de très mauvaise qualité.

Pour les politiques en ligne, on remarque sur la figure 1.2(b) que les politiques FIFO et LPT sont dominées. Les 20 premiers utilisateurs ont des débits de mauvaise qualité avec ces politiques en ligne tandis que les 20 suivants ont un débit comparable avec les politiques hors-ligne.

Lorsque l'utilisation des machines s'effectue par soumission interactive des utilisateurs, la durée des tâches est souvent inconnue à leur arrivée, cependant compte-tenu de l'utilisateur ou de son groupe une statistique d'utilisation aide à la décision de placement des tâches. Par exemple l'ordonnanceur MAUI [Jackson et al., 2001] maintient une file d'attente des tâches disponibles triées selon leur priorité. La priorité d'une tâche est calculée en sommant différents bonus de priorité selon le temps passé dans la file d'attente, le groupe de la tâche, son utilisateur, la consommation de son groupe et de son utilisateur, etc. L'administrateur peut attribuer des coefficients à chacun de ces bonus.

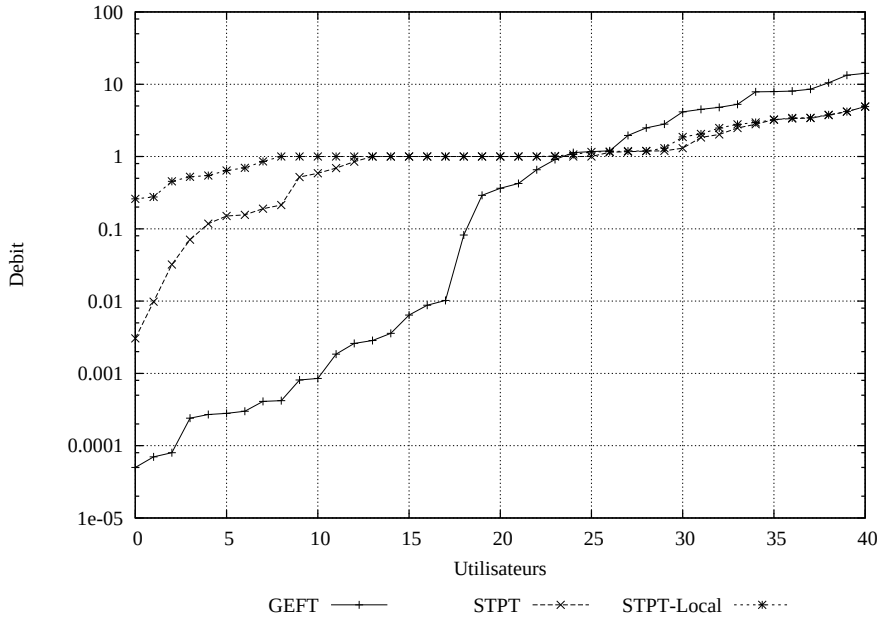
Nous nous intéressons au problème de planification de production. Les grands centres de calcul établissent des calendriers de production : chaque année les utilisateurs soumettent un projet scientifique évalué par un conseil scientifique. Ce conseil scientifique formule des recommandations qui vont permettre d'effectuer les allocations d'heures sur les machines de calcul référencées par le projet. Nous supposerons dans la suite que les caractéristiques des tâches sont connues lors de l'établissement des plages horaires de ce calendrier de production.

Pour finir, nous présentons un exemple de problème où tout les ordonnancements de liste ont été générés, (voir la figure 1.4.2). Les solutions qui maximisent les débits des utilisateurs pour le Leximin et NBS (voir chapitre suivant) sont indiquées.

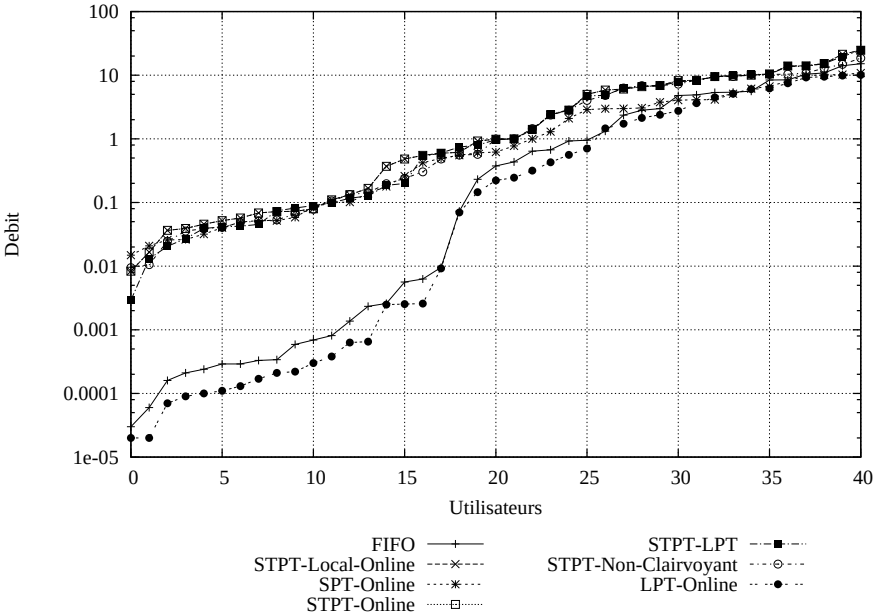
1.4.1 Conclusions

Résumons les résultats de ce chapitre. Nous avons tout d'abord considérer les différents aspects de la notion d'équité. Nous avons ensuite proposer d'appréhender le problème de l'ordonnancement de tâches de plusieurs utilisateurs sous l'angle de l'arbitrage de ressources, il en découle que cet arbitrage se doit d'être équitable.

Le contexte de l'étude a été présenté (ordonnancement sur grille de calcul). Un état de l'art des différentes approches sur l'équité a été présenté et nous avons porté un examen critique de ces propositions.



(a) Évaluation de différentes politiques hors-ligne sur les logs [Medernach, 2005] d'un clusters avec 5000 tâches et 50 machines.



(b) Évaluation de différentes politiques en ligne sur ces mêmes logs.

Utilisateur	Durée
1	1
1	9
1	10
1	19
2	18
2	3
2	7
2	16

Tab. 1.4 Problème considéré avec 2 utilisateurs, toutes les tâches sont disponibles à la date 0.

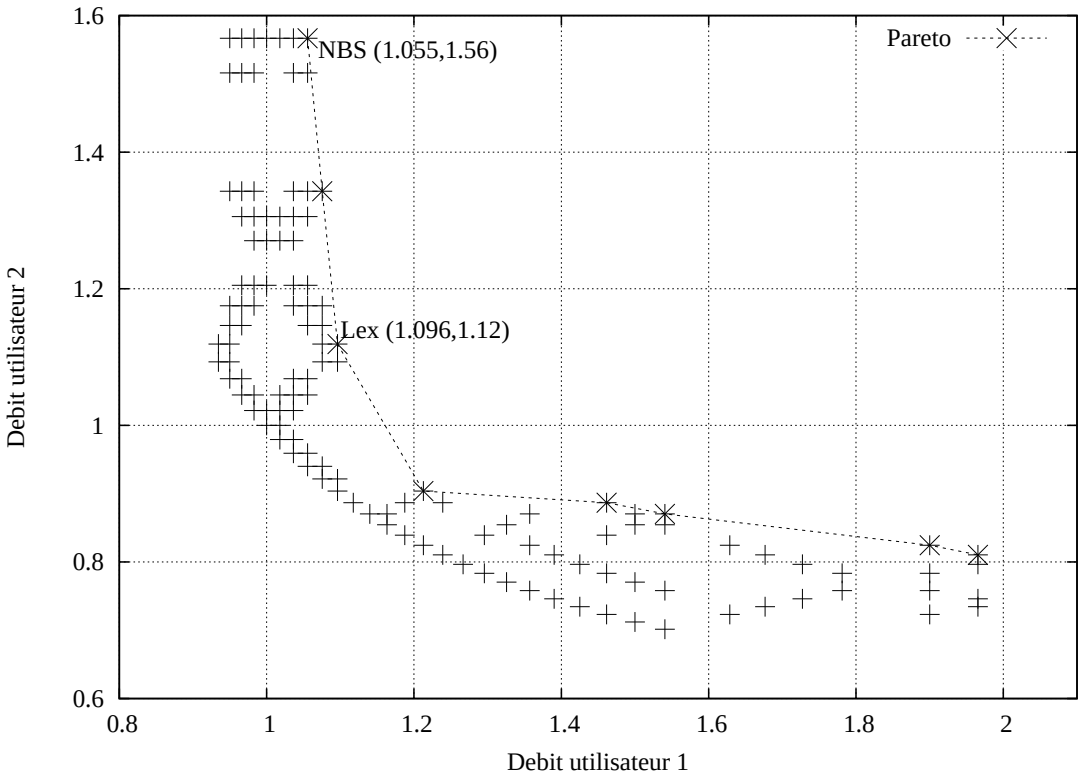


Fig. 1.4.2 Débits obtenus pour tous les ordonnancements possibles sur 2 machines avec 2 utilisateurs ayant 4 tâches chacun. L'ensemble de Pareto est représenté ainsi que les solutions NBS et Leximin.

Plutôt qu'une approche égalitariste ou utilitariste de l'équité nous nous sommes orienté vers une notion d'équité relative à un principe de comparaison dont nous détaillerons les principes dans le prochain chapitre.

Nous allons considérer l'ordonnancement sous l'angle de l'allocation de ressources et de la gestion de ressources au cours du temps, tout en satisfaisant au mieux un ensemble de critères basés sur l'équité.

Chapitre 2

Équité et optimisation multi-utilisateurs

> J'estime que la plus grande part des misères de l'humanité est due aux estimations erronées qu'elle fait quant à la valeur des choses.
>
> Benjamin Franklin

Ce chapitre présente les notions d'optimisation multi-utilisateurs et d'équité sous un angle multi-critères. Dans un premier temps nous décrivons les notations et propriétés utilisés, dans un premier temps dans le cadre de critères quelconques, puis dans le cas de critères comparables.

2.1 Optimisation multi-critères déterministe avec critères quelconques

L'ensemble des solutions est noté $\mathcal{S}$.

Définition 2.1 (Espace des solutions)**.** *Un espace des solutions $\mathcal{S}$ est l'ensemble des solutions d'un problème : l'ensemble des éléments qui vérifient les contraintes du problème.*

Définition 2.2 (Espace d'évaluation)**.** *Un espace d'évaluation $(E, \prec)$ est un ensemble E muni d'une relation d'ordre éventuellement partielle $\prec$.*

Définition 2.3 (Fonction d'évaluation)**.** *Une fonction qui associe à chaque solution de $\mathcal{S}$ un élément d'un espace d'évaluation $(E, \prec)$ est une fonction d'évaluation. Par extension, on note l'ordre $\prec_f$ pour comparer les solutions : $s_1 \prec_f s_2 \Leftrightarrow f(s_1) \prec f(s_2)$.*

Un critère $\prec_f$ est ainsi défini par la fonction d'évaluation f, par abus de langage on parle aussi de critère f pour désigner le critère $\prec_f$.

Définition 2.4 (Critère numérique ou utilité)**.** *Un critère numérique (on parle aussi d'utilité) est un ordre associé à une fonction d'évaluation $f : \mathcal{S} \mapsto \mathbb{R}$ appelée fonction objectif, de sorte que $X \prec_f Y \Leftrightarrow f(X) \leq f(Y)$.*

On se place dans le cas de problèmes de maximisation, le cas de problèmes de minimisation étant symétrique. L'optimisation multi-critères considére un ensemble $\mathcal{C}$ fini ou dénombrable de critères.

Définition 2.5 (Critère et relation d'équivalence selon un critère)**.** *Un critère $\prec$ est un ordre total entre éléments de l'ensemble des solutions $\mathcal{S}$ muni de la relation d'équivalence $\equiv$. On a*

$$a \prec b \text{ et } b \prec a \Leftrightarrow a \equiv b$$

On associe à toute solution son vecteur d'évaluation :

Définition 2.6 (Vecteur d'évaluation). *Considérons un ordre entre les critères de l'ensemble* $\mathcal{C} = \{\prec_{f_1}, \prec_{f_2}, \ldots\}$. *On associe à chaque solution* S, *un vecteur* $V(S) = [f_1(S), f_1(S), \ldots]$ *appelé vecteur des critères.*

Il faut noter qu'il existe des critères qui ne sont pas des critères numériques [Debreu, 1954] :

Définition 2.7 (Ordre lexicographique). *Soit un ensemble ordonné de critères* $\mathcal{C} = \{\prec^1, \prec^2, \ldots\}$ *fini ou dénombrable. L'ordre lexicographique sur* $\mathcal{C}$ *est noté* $\prec_{Dict,\mathcal{C}}$ *par analogie avec l'ordre habituel du dictionnaire. On a :*

$$S \prec_{Dict,\mathcal{C}} S' \Leftrightarrow \forall i, S \equiv_i S' \text{ ou } \exists i, \forall 1 \leq j < i, S \equiv_j S' \text{ et } S \not\equiv S' \text{ et } S \prec^i S'$$

L'ordre lexicographique est similaire à l'ordre naturel entre les nombres réels. En effet, pour comparer deux nombres, on compare successivement les chiffres de la représentation de ces nombres en commençant par les chiffres dont la position est de poids le plus élévé. Ainsi, lorsque les critères prennent des valeurs dans un ensemble fini, on peut simplement représenter l'ordre lexicographique par une fonction scalaire analogue à la représentation des nombres.

Lorsque chaque critère prend ses valeurs dans $\mathbb{R}$, l'ordre lexicographique n'est alors pas représentable par une fonction d'utilité. Cette remarque n'est pas insignifiante : le principe de l'école de pensée utilitariste [Bentham, 1823] stipule que tout acte peut se justifier dans la mesure où il maximise le "bonheur total" de toutes les personnes affectées par cet acte. Cette philosophie n'est pas unanimement admise, en effet peu de personnes acceptent l'idée de justifier la maltraitance ou l'esclavage pour le bien-être d'autres personnes par exemple.

La philosophie utilitariste prétend ramener avec pragmatisme toute question de principe à une maximisation d'une seule fonction scalaire censé représenter un état de bonheur global, mais cependant fictif. On voit donc sur ce simple exemple que ce point de vue est non seulement réducteur [1] mais peut entrainer à des compromis de principe douteux.

Définition 2.8 (Dominance). *Une solution* S *domine une solution* S' *si pour tout critère* $\prec \in \mathcal{C}$ *considéré, on a* $S' \prec S$. *Si de plus* $S \neq S'$ *on dit que* S *domine strictement* S'.

Définition 2.9 (Ordre partiel de dominance). *Notons* $\prec_c$, *l'ordre partiel de comparaison critère par critère (ou coordonnée par coordonnée si les critères sont numériques) :*

$$S \prec_c S' \Leftrightarrow \forall \prec \in \mathcal{C}, S \prec S'$$

Définition 2.10 (Optimalité de Pareto). *Une solution* S *est dite* Pareto optimale *si elle est non strictement dominée dans l'ensemble des solutions. De façon équivalente,* S *est un élément maximal de l'ordre partiel de dominance.*

En général, l'ensemble des solutions Pareto optimale n'est pas un singleton. Lorsqu'une seule solution doit être selectionnée selon différents critères, il ne faut considérer que les solutions Pareto optimales. En effet, ces solutions excluent tout sous-dimensionnement (ou "gaspillage") au sens où ne sont pas retenues les solutions qui sont pas améliorables pour tous les critères. De plus, par définition il est impossible d'améliorer un critère d'une solution Pareto optimale sans en détériorer un autre.

1. Tout comme le fait d'aggréger des critères en optimisation multi-critères peut aussi être réducteur.

Exemple

Considérons l'exemple suivant : 6 experts donnent leur avis sur un ensemble $\{A, B, C, D, E, F, G\}$ de choix. Un seul choix doit être pris parmi cet ensemble.

Expert	Avis
1	$A = B = E = F \prec D \prec C = G$
2	$A = B = E \prec C = D = F = G$
3	$A = B = F \prec C = D \prec E = G$
4	$A = F \prec B = C = D \prec E = G$
5	$A = B = C = D = F \prec E = G$
6	$A = D = F \prec B = C = E = G$

Tab. 2.1 Avis de 6 experts sur plusieurs choix $\{A, B, C, D, E, F, G\}$

Le choix A est dominé car tous les experts lui préférent le choix B. Le choix G domine l'ensemble car il est préféré à tous les autres par tous les experts. Supposons maintenant que ce choix est impossible à prendre. L'ensemble de Pareto est l'ensemble des éléments maximaux pour l'ordre de dominance. Sans G c'est l'ensemble $\{C, E\}$. On voit ici se profiler les difficultés inhérentes aux systèmes de vote pour le choix d'un unique candidat.

Définition 2.11 (Suppression d'une composante d'un vecteur)**.** *L'opération de suppression de la composante i associe au vecteur $[x_1, \dots]$ le vecteur $[x_1, \dots, x_{i-1}, x_{i+1}, \dots]$.*

Définition 2.12 (Adjonction d'une valeur à vecteur)**.** *L'opération d'adjonction d'une valeur a associe au vecteur $[x_1, \dots]$ le vecteur $[a, x_1, \dots]$. On note $[a, x_1, \dots] = a \oplus [x_1, \dots]$.*

Théorème 2.1 (Composantes liées)**.** *Si $\forall a, b \in E, a_j \geq b_j \Rightarrow a_i \geq b_i$ alors on peut supprimer la composante i pour calculer le front de Pareto de E.*

Démonstration. Notons E' l'ensemble des éléments de E auquel on a supprimé la composante i.

Soit X un élément non dominé de E et X' auquel on a supprimé la composante i. Il n'existe pas de $Y' \in E'$ tel que $X' \prec_c Y'$ car comme $Y' \in E'$ il existe $Y \in E$ tel que $Y = y_i \oplus Y'$ et comme $y_j \geq x_j$, on a par hypothèses $y_i \geq x_i$ soit $X \prec_c Y$, ce qui contredit le fait que X est un élément non dominé de E.

Réciproquement, soit X' un élément non dominé de E'. Alors il existe plusieurs X tels que X' est obtenu par suppression de la composante i de X. Fixons X à celui qui maximise la composante x_i, alors X est non dominé dans E. En effet, supposons que $X \prec_c Y$ alors $X' \prec_c Y'$ or comme X' est un élément non dominé de E', on a $X' = Y'$. Et puisque x_i est maximal, on a $X = Y$. ∎

Le concept de solution permet de définir ce qu'on entend par solution optimale dans un problème multi-critères particulier.

Définition 2.13 (Ensemble des valeurs optimales selon un concept de solution)**.** *L'ensemble des valeurs des solutions optimales selon le concept de solution $(p, \theta, \preceq)$ est l'ensemble des éléments maximaux de $\{\theta(V(S)) | S \in \mathcal{S}\}$ pour l'ordre $\preceq$.*

Définition 2.14 (Ensemble des solutions optimales selon un concept de solution)**.** *L'ensemble des solutions optimales selon un concept de solution $(p, \theta, \preceq)$ sont les solutions $S \in \mathcal{S}$ telles que $\theta(V(S))$ soient dans l'ensemble des valeurs optimales. On note cet ensemble $OPT(p, \theta, \preceq)$.*

2.1.1 Optimisation multi-critères déterministe avec critères comparables

Il est naturel de considérer que chaque utilisateur possède un critère pour évaluer la qualité d'une solution. La notion d'équité intervient lorsqu'il est possible de comparer le niveau de service obtenu par les différents utilisateurs, dans le but de garantir un service qui ne pénalise aucun utilisateur. Dans les problèmes d'équité, le critère est appelé un critère de *comparaison.*

Définition 2.15 (Critères numériques comparables [Deschamps et Gevers, 1978])**.** *Deux critères numériques* $\prec_f$ *et* $\prec_g$ *sont des* critères comparables *si et seulement s'il est possible pour tout* X *de comparer* $f(X)$ *et* $g(X)$*.*

En effet, toutes les fonctions numériques ne sont pas pour autant comparables, dès lors que chacune s'exprime dans des unités différentes comme le poids ou la dimension.

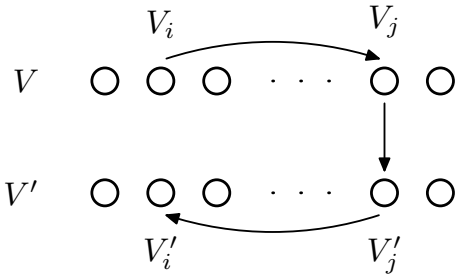
On considère désormais N critères numériques comparables.

Définition 2.16 (Concepts de solution [Ehrgott, 1996, 2000])**.** *Un concept de solution est un triplet* $(p, \theta, \preceq)$ *où*

- p *est un entier positif.*
- θ *est une fonction* $\mathbb{R}^N \mapsto \mathbb{R}^p$
- $\preceq$ *est un ordre partiel de* $\mathbb{R}^p$

Définition 2.17 (Vecteur trié d'un vecteur)**.** *A tout vecteur* V *dont les composantes sont comparables on associe un vecteur* $\vec{V}$ *dont les composantes sont triées : c'est à dire tel que* $\vec{V}[1] \leq \vec{V}[2] \leq \dots$ *avec* $\vec{V}_i = V_{\sigma(i)}$ *où* σ *est une permutation.*

Théorème 2.2 (Tri et dominance)**.** *Si* $V \prec_c V'$ *alors* $\vec{V} \prec_c \vec{V}'$*.*



Démonstration. Soit V_i tel que $V_i = \min(V)$ et V'_j avec $V'_j = \min(V')$, alors on a $V_i = \min(V) \leq V_j \leq V'_j = \min(V')$ et $\vec{V}_1 \leq \vec{V'}_1$.

Or $V_j \leq V'_j = \min(V') \leq V'_i$ et $V_i = \min(V) \leq V_j \leq V'_j$, donc si on intervertit V'_i avec V'_j dans V' on obtient V'' avec $V \prec_c V''$ et $V'_i = \min(V')$, $V_i = \min(V)$. On peut alors poursuivre le raisonnement en supprimant la i^e coordonnée pour obtenir le résultat. ∎

2.1.1.a Garantie de performance

On recherche une mesure de l'équité qui est un ordre sur les vecteurs de critères dont les indices sont les utilisateurs. Si l'ensemble des critères sont numériques et comparables, on peut alors définir :

Définition 2.18 (Ordre induit par $\min$ ou $\max$ (Min Ordering/Max Ordering)). *L'ordre d'agrégation des critères induit par* $\min$ *est noté* $\prec_{\min}$. *Cet ordre compare deux ensembles de critères comparables en comparant le pire des cas :*

$$S \prec_{\min} S' \Leftrightarrow \min V(S) \leq \min V(S')$$

De même l'ordre induit par $\max$ *est noté* $\prec_{\max}$ *avec :*

$$S \prec_{\max} S' \Leftrightarrow \max V(S) \geq \max V(S')$$

On parle aussi de critère min (et respectivement max).

Définition 2.19 (Garantie de service). *La garantie de service désigne la capacité à fournir et à assurer un niveau de service conforme à des exigences selon un critère donné.*

Soit Q_i *le niveau de service de l'utilisateur* U_i, *alors la garantie de service s'exprime par la contrainte* $V_i \geq Q_i$.

Le fait de garantir un niveau de service minimal conduit à la notion d'équité au sens faible :

Définition 2.20 (Équité au sens faible). *L'équité au sens faible consiste à assurer un niveau de service maximal identique pour tous les utilisateurs : une solution est équitable au sens faible si son vecteur d'évaluation* V *maximise* $\min_i V_i$.

Notons bien que les solutions équitables au sens faible ne sont pas toutes non dominées : si un élément est maximal pour $\prec_{\min}$ il n'est pas pour autant Pareto optimal. Par exemple, considérons l'ensemble suivant : $\{[1,8],[3,2],[4,2]\}$, l'élément $[3,2]$ est maximal pour $\prec_{\min}$ et est strictement dominé par $[4,2]$.

L'ordre induit par $\min$ n'est pas une fonction de comparaison complétement satisfaisante pour l'équité : cet ordre impose une qualité de service minimum maximale (dans le sens de service garanti) mais n'impose pas forcément l'équité envers tous les autres utilisateurs.

2.1.2 Agrégation de critères

Pour comparer les solutions dans un problème multi-critères une méthode possible consiste à agréger les différents critères afin de se ramener au cas d'un seul critère numérique.

Définition 2.21 (Sommes pondérées ordonnées (OWA) [Yager, 1988; Roy, 2007]). *Soient* $w_1, \ldots, w_N$ *des pondérations :* $\forall w_i, 0 \leq w_i \leq 1$ *et* $\sum_{i=1}^{N} w_i = 1$. *Le critère* $OWA(V)$ *pour cette pondération est défini par :*

$$OWA(V) = \sum_{i=1}^{N} w_i \vec{V}_i$$

$OWA_{\min}(V)$ *[Dubois et Prade, 1996] pour cette pondération est défini par :*

$$OWA_{\min}(V) = \min_i w_i \vec{V}_i$$

On peut noter que la famille des sommes pondérées ordonnées contient entre autre le critère $\min$ avec $w_1 = 1, w_2 = \cdots = w_N = 0$, le critère $\max$ avec $w_N = 1, w_1 = \cdots = w_{N-1} = 0$, la moyenne avec $w_1 = \cdots = w_N$ et la médiane avec $w_{N/2} = 1, \forall i \neq N/2, w_i = 0$.

Il y a un problème commun à toutes les agrégations de critères : il existe des lignes de niveau (iso-surfaces) où les solutions sont considérées comme équivalentes. Ces ensembles de même niveau ne sont pas toujours satisfaisants, comme nous l'avons vu pour l'équité au sens faible.

2.1.2.a L'ordre Leximin et ses propriétés

En général les critères Max-Min [Jaffe, 1981] sont appliquées itérativement en maximisant les minimums les uns après les autres. Un raffinement de l'ordre $\min$ consiste à poursuivre sur le second critère le moins avantagé. Si on continue ainsi de suite on obtient un ordre appelé *Leximin* :

Définition 2.22 (Ordre Leximin [Sen, 1970; Rawls, 1971; Megiddo, 1977; Deschamps et Gevers, 1978; Kolm, 1997; Yager, 1997; Kostreva et al., 2004])**.** *L'ordre Leximin* $\prec_{Lex}$ *est défini par :*

$$S \prec_{Lex} S' \Leftrightarrow \vec{V}(S) \prec_{Dict} \vec{V}(S')$$

Le critère d'équivalence pour l'ordre Leximin est l'équivalence à une permutation près du vecteur des critères, ie cet ordre est insensible à l'ordre considéré entre les critères.

De façon équivalente on peut définir le Leximin par :

Lemme 2.3. $X^i \prec_{Lex} X \Leftrightarrow \vec{X}_i \prec_{Dict} \vec{X}$

Il est possible de découper l'espace d'évaluation en plusieurs zones :

Soit σ une permutation de $\mathfrak{S}_N$ et C_σ l'ensemble des points $x \in \mathbb{R}^N$ tels que $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(N)}$. Cet ensemble est un cône ; en effet, si un point x appartient à cet ensemble $\lambda \in \mathbb{R}, \lambda x \in C_\sigma$.. De plus, ces ensembles C_σ recouvrent l'espace : $\mathbb{R}^N = \bigcup_{\sigma \in \mathfrak{S}_N} C_\sigma$ car chaque point x fait parti de l'ensemble C_σ où σ est telle que $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(N)}$.

Maximiser le Leximin dans C_σ revient à maximiser l'ordre lexicographique sur le vecteur $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(N)}$; ce qui revient donc à maximiser la coordonnée $\sigma(1)$, puis $\sigma(2)$, etc, jusqu'à $\sigma(N)$. Comme ces cônes recouvrent l'espace, maximiser le Leximin revient donc à maximiser l'ordre lexicographique pour une permutation σ particulière.

La façon de comparer deux vecteurs ressemble à la façon de comparer deux nombres en représentation décimale, on commence par comparer les unités les plus importantes et si elles sont égales on continue sur les suivantes, etc. On peut ainsi faire l'analogie entre un vecteur pour l'ordre Leximin et l'ordre sur les nombres réels lorsque l'ensemble est fini.

Proposition 2.1. *Le leximin est un ordre complet à permutation des composantes près.*

Démonstration. $\prec_{Lex}$ est une relation d'ordre sur les listes triées car

- $\prec_{Lex}$ est réflexive car $X \prec_{Lex} X$
- $\prec_{Lex}$ est antisymétrique. Soit X et Y tels que $X \prec_{Lex} Y$ et $Y \prec_{Lex} X$ alors soit $X = Y$, soit $\exists j_1, \forall i < j_1, x_i = y_i$ et $x_{j_1} < y_{j_1}$ et $\exists j_2, \forall i < j_2, x_i = y_i$ et $x_{j_2} < y_{j_2}$. Dans ce cas on a une contradiction si on prend $i = \min(j_1, j_2)$. Donc $X = Y$.
- $\prec_{Lex}$ est transitive. Si $X \prec_{Lex} Y$ et $Y \prec_{Lex} Z$ alors soit $X = Y$ ou $Y = Z$ et on a $X \prec_{Lex} Z$, soit $\exists j_1, \forall i < j_1, x_i = y_i$ et $x_{j_1} < y_{j_1}$ et $\exists j_2, \forall i < j_2, y_i = z_i$ et $y_{j_2} < z_{j_2}$. Si $j_1 \leq j_2$ et en prenant $j = j_1$ on a $x_{j_1} < y_{j_1} = z_{j_1}$, ou bien $j_2 \leq j_1$ et en prenant $j = j_2$ on a $x_{j_2} = y_{j_2} < z_{j_2}$.

■

2.1.3 Propriétés du Leximin

Définition 2.23 (Propriété de normalisation [Ehrgott, 1996, 2000] d'un concept de solution)**.** *Le concept de solution* $(p, \theta, \leq)$ *vérifie la propriété de normalisation si et seulement si lorsque* $N = 1$ *il est équivalent à* $(1, id, \leq)$.

Définition 2.24 (Contraintes sur les critères). *Soit $\mathcal{S}' \subset \mathcal{S}$ et $C = \{(i_1, y^1), \ldots, (i_q, y^q)\}$. L'ensemble $\mathcal{S}'$ sous contraintes C est l'ensemble des $S \in \mathcal{S}'$ telles que $\forall i_a \in \{i_1, \ldots, i_q\}, V(S)_{i_a} = y^a$.*

Définition 2.25 (Propriété de réduction [Ehrgott, 1996, 2000] d'un concept de solution). *Le concept de solution $(p, \theta, \preceq)$ vérifie la propriété de réduction si et seulement si l'ensemble des solutions optimales de $\mathcal{S}$ sous contraintes C est l'ensemble des solutions optimales sous contraintes C de $\mathcal{S}$.*

Définition 2.26 (Régularité [Ehrgott, 1996, 2000] d'un concept de solution). *Un concept de solution est régulier si et seulement si l'ensemble des solutions optimales selon ce concept de solution est inclus dans l'ensemble des solutions optimales selon $(1, \min, \prec_c)$.*

[Ehrgott, 1996, 2000] montrent alors le théorème suivant :

Théorème 2.4 (Caractérisation du Leximin). *Si un concept de solution $(p, \theta, \preceq)$ vérifie les propriétés de normalisation, de réduction et de régularité alors $(p, \theta, \preceq) = (N, V \mapsto \vec{V}, \prec_{Lex})$ et réciproquement.*

Définition 2.27 (Vecteur des sommes partielles). *Soit $V = [V_1, \ldots, V_N]$, on note $\nearrow V$ le vecteur des sommes partielles de V avec $\nearrow V = [V_1, V_1 + V_2, \ldots, V_1 + \cdots + V_N]$.*

Proposition 2.2. $V \prec_{Dict} V' \Leftrightarrow \nearrow V \prec_{Dict} \nearrow V' \Leftrightarrow \nearrow V \prec_{Lex} \nearrow V'$

Démonstration. En effet, comme $\nearrow V$ est un vecteur trié, le premier élément est le minimum. Itérativement sur les indices on obtient le résultat suivant : soit les éléments sont égaux et on poursuit sur l'indice suivant, soit celui de V' ou de $\nearrow V'$ est strictement supérieur et on a alors l'autre inégalité. ∎

Proposition 2.3. *Le Leximin n'est généralement pas représentable par une fonction scalaire : il n'existe pas de fonctions $f : \mathbb{R}^N \mapsto \mathbb{R}$ telle que $X \prec_{Lex} Y \Leftrightarrow f(X) \leq f(Y)$.*

Démonstration. Comme $V \prec_{Lex} V' \Leftrightarrow \vec{V} \prec_{Dict} \vec{V'}$ et que $\vec{V} = \nearrow \Delta\vec{V}$, avec $\Delta V = [V_1, V_2 - V_1, \ldots, V_N - V_{N-1}]$, on a $V \prec_{Lex} V' \Leftrightarrow \Delta\vec{V} \prec_{Dict} \Delta\vec{V'}$. Comme $\prec_{Dict}$ n'est pas représentable par une fonction scalaire, on a le résultat. ∎

L'ordre Leximin ne s'obtient donc pas par une approche utilitariste où l'on cherche à maximiser une fonction d'utilité scalaire, comme la somme. Ainsi par exemple, on ne peut pas obtenir le Leximin par combinaisons linéaires.

En outre, il est difficile d'imaginer une notion d'approximation entre deux vecteurs qui soit conforme au Leximin. Par exemple pour définir une approximation dans le cas d'un objectif scalaire, on utilise un coefficient de proportionnalité : pour un objectif scalaire une solution a un coefficient d'approximation ρ si le rapport entre la valeur de la solution et la valeur de la solution optimale est en dessous de ρ. Mais dans le cas du Leximin, ces approches ne sont pas satisfaisantes. En effet, comparer des vecteurs pondérés avec le Leximin est le plus souvent équivalent à ne comparer que le minimum des coordonnées.

Mais si on s'intéresse à un sous-ensemble fini de $\mathbb{R}^N$ alors les ordres OWA peuvent représenter le Leximin ; néanmoins cela nécessite d'avoir des coefficients linéaires trés hétérogènes et donc des instabilités numériques dans la résolution.

Maximiser le Leximin sur les pavés de $\mathbb{R}^n$: "Water filling"

Le problème de maximiser l'ordre Leximin sur un pavé de $\mathbb{R}^n$ est un problème classique que l'on retrouve dans l'attribution de flots réseaux [Bertsekas et Gallager, 1987; Boudec, 2000; Nace et al., 2006] :

Définition 2.28 (Pavés de $\mathbb{R}^N$ sous contrainte de somme)**.** *Soient X, Y des vecteurs de $\mathbb{R}^N$ avec $X \prec_c Y$. Notons $([X,Y],M)$ l'ensemble des vecteurs Z tels que :*
- $\sum_{k=1}^{N} Z_k \leq M$
- *et $X \prec_c Z \prec_c Y$.*

Définition 2.29 (Vecteur maximum pour le Leximin)**.** *Un vecteur maximum pour le Leximin dans $([X,Y],m)$ est noté $Eq([X,Y],M)$. Tous les autres vecteurs maximaux pour le Leximin s'obtiennent à partir de celui-ci par permutation des utilisateurs.*

Si $\sum X > M$ $([X,Y],m)$ est l'ensemble vide. Si $\sum Y \leq M$ alors le maximum est Y. Supposons maintenant que $\sum X \leq M \leq \sum Y$.

Lemme 2.5. *Il existe $\alpha \in \mathbb{N}$ et une partition des utilisateurs en 4 ensembles, E_1, E_2, E_3, E_4, avec :*
- $k \in E_1 \Rightarrow Eq([X,Y],m)_k = Y_k$
- $k \in E_2 \Rightarrow Eq([X,Y],m)_k = \alpha$
- $k \in E_3 \Rightarrow Eq([X,Y],m)_k = \alpha + 1$
- $k \in E_4 \Rightarrow Eq([X,Y],m)_k = X_k$

Démonstration. Notons $Z_k = (Eq([X,Y],m))_k$.

Supposons qu'il existe k avec $X_k < Z_k < Y_k$ et Z_k minimum. Soit $\alpha = Z_k$.

Soit k' tel que $X_{k'} < Z_{k'} < Y_{k'}$. Si $Z_{k'} \geq Z_k + 2$, alors ré-allouer une machine de l'utilisateur $U_{k'}$ vers U_k conduit à un vecteur dans $([X,Y],M)$ qui est strictement supérieur pour $\prec_{Lex}$, ce qui est contradictoire.

Ainsi, les indices k appartiennent à un des 4 ensembles ci-dessus.

■

α est aussi désigné comme le *niveau d'eau*.

Calcul du niveau d'eau

D'après le thèorème précédent, on sait que $Eq([X,Y],m) = Z$ avec

$$Z_k = \min(Y_k, \max(X_k, \alpha)) + \epsilon, \epsilon \in \{0,1\}$$

Soit la fonction croissante $F(x) = \sum_k \min(Y_k, \max(X_k, x))$, comme $\sum X \leq M \leq \sum Y$, M a un réciproque par F. Notons a ce réciproque, alors $\alpha = \lfloor x \rfloor$. En effet, $F(\lfloor x \rfloor) \leq F(x) = M \leq F(\lceil x \rceil)$. Or le calcul de la réciproque de M par F peut s'effectuer en $O(log(N))$ par dichotomie sur F, qui est croissante.

Exemple

La figure 2.1.1 montre un exemple avec $X = [1,5,1,1,5,3,1]$, $Y = [7,9,2,6,7,5,2]$ et $m = 38$, on a $\alpha = 3$ et $Eq([X,Y],m) = [3,5,2,3,5,4,2]$ à une permutation près.

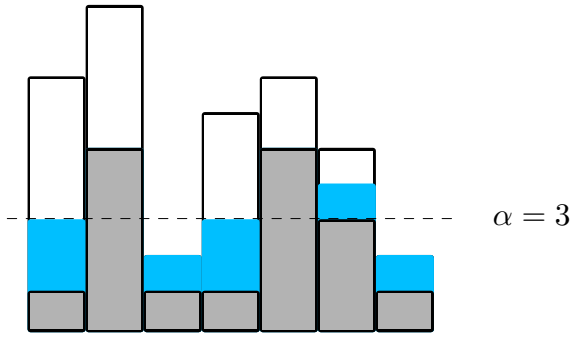


Fig. 2.1.1 Exemple de water-filling.

Exemple avec pondérations

Montrons un exemple avec deux principes d'évaluations différents. Soit le vecteur des demandes $V = [1, 2, 4, 6, 20, 24, 30]$ avec 64 machines à répartir. Le vecteur d'allocation le plus équitable est alors $A = [1, 2, 4, 6, 17, 17, 17]$. Supposons que le principe d'évaluation doit tenir compte des pondérations suivantes : $W = [1.0, 1.0, 4.0, 4.0, 6.0, 7.5, 12.0]$, de sorte que le principe d'évaluation compare maintenant les A_h/W_h. On cherche donc à ce que le vecteur d'évaluation $E = [A_1/W_1, \dots]$ soit le plus équitable possible. Dans ce cas le vecteur le plus équitable est de la forme $E_h = A_h/W_h = \min(V_h/W_h, \alpha)$. Ainsi si $V_h \leq \alpha W_h$ on a $A_h = V_h$, sinon $A_h = \alpha W_h$, soit $A_h = \min(V_h, \alpha W_h)$. Ici on trouve $\alpha = 2.0$ avec le vecteur des allocations suivant $A = [1, 2, 4, 6, 12, 15, 24]$.

Ainsi ce principe d'évaluation pondéré couplé avec le Leximin peut être utilisé lorsqu'on cherche la proportionnalité.

2.1.3.a Cas particulier : $X = 0$

Lorsque la borne inférieure est nulle, notons $Eq(V, m)$ le vecteur $Eq([0, V], m)$. Observons que l'on retrouve la règle de Maïmonides [Young, 1994; Thomson, 2007] qui donne une méthode de répartition de biens lors d'un conflit :

Définition 2.30 (Règle de Maïmonides (12e siècle))**.** *Soit M la somme totale à répartir et V_i la demande de l'utilisateur i, alors la règle de Maïmonides stipule qu'il faut répartir de la façon suivante : il existe un α tel que la part de l'utilisateur i est $A_i = \min(\alpha, V_i)$ avec $M = \sum_i A_i$.*

Contrairement à une répartition proportionnelle, la règle de Maïmonides n'encourage pas la surenchère et aucun participant ne peut monopoliser les ressources.

Les propriétés suivantes seront utilisées par la suite :

Lemme 2.6 (Propriétés de $Eq(V, M)$)**.** *On a :*
- *Si $V \prec_c W$, alors $Eq(V, M) \prec_{Lex} Eq(W, M)$.*
- *Si $M_1 \leq M_2$, alors $Eq(V, M_1) \prec_{Lex} Eq(V, M_2)$.*

Démonstration. Comme $V \prec_c W$, $([0, V], M) \subset ([0, W], M)$ et comme $Eq(V, M) \in ([0, V], M)$, $Eq(V, M) \prec_{Lex} Eq(W, M)$. La seconde propriété est une conséquence directes du lemme 2.5. ∎

2.1.3.b L'ordre de Majorisation

L'ordre de majorisation a des propriétés qui le rendent intéressant du point de vue de l'équité [Hardy et al., 1929; Bhargava et al., 2001; Goel et al., 2001b,a; Goel et Meyerson, 2006; Kyselovà et al., 2007].

Définition 2.31 (Ordre partiel de Majorisation [Schur, 1923]). *Soient $X, Y \in \mathbb{R}^n$ avec*

$$\forall 1 \leq k \leq n, \sum_{i=1}^{k} \vec{X}_i \leq \sum_{i=1}^{k} \vec{Y}_i \text{ ou } \nearrow \vec{X} \prec_c \nearrow \vec{Y}$$

Alors on dit que X est faiblement majoré par Y et on note $X \prec_{Maj} Y$.
Si de plus $\sum_{i=1}^{n} \vec{X}_i = \sum_{i=1}^{n} \vec{Y}_i$, alors on dit que X est majoré au sens fort par Y et on note $X \prec_{MAJ} Y$.

Prenons un exemple : $[\frac{1}{n}, \ldots, \frac{1}{n}] \prec_{Maj} [\frac{1}{i}, \ldots, \frac{1}{i}, 0, \ldots, 0]$
En Economie cet ordre s'appelle aussi d'ordre de Lorenz [Arnold, 1987]. La majorisation est un ordre partiel sur $\mathbb{R}^n$ muni de $\equiv$. Le résultat suivant a été démontré par [Hardy et al., 1934] :

Théorème 2.7. *$X \prec_{MAJ} Y$ si et seulement si il existe une matrice A doublement stochastique telle que $X = AY$.*

Pour la mesure de l'équité la majorisation généralise les métriques de Jain, la variance, etc en remarquant que si une fonction numérique doit mesurer l'équité alors cette fonction doit au moins être symétrique (invariante par permutation de coordonnées) et concave (car $[x_1, \ldots, x_N]$ est moins équitable que $[\frac{\sum_{i=1}^{N} x_i}{N}, \ldots]$). [Goel et al., 2001a; Bhargava et al., 2001; Goel et Meyerson, 2006] proposent d'utiliser l'ordre de majorisation grâce au résultat suivant :

Définition 2.32 (Inégalité de Hardy-Littlewood-Polya-Karamata). *Si $X \prec_{MAJ} Y$ alors pour toute fonction concave F on a $\sum_{i=1}^{n} F(X_i) \geq \sum_{i=1}^{n} F(Y_i)$*

Il se trouve que l'ordre de majorisation est un sous-ordre du Leximin et un sur-ordre de $\prec_c$:

Proposition 2.4. *Soient x, y avec $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, alors*
- *$x \prec_c y \Rightarrow x \prec_{Maj} y$ ($\prec_c$ est un sous-ordre de la Majorisation)*
- *$x \prec_{Maj} y \Rightarrow x \prec_{Lex} y$ (La Majorisation est un sous-ordre du Leximin)*

Démonstration. Comme $\vec{x}_1 \leq \vec{y}_1$, si cette inégalité est stricte alors $\vec{x} \prec_{Lex} \vec{y}$, sinon $\vec{x}_1 = \vec{y}_1$ et comme $\vec{x}_1 + \vec{x}_2 \leq \vec{y}_1 + \vec{y}_2$, on a $\vec{x}_2 \leq \vec{y}_2$. Récursivement on obtient que $x \prec_{Maj} y \Rightarrow x \prec_{Lex} y$. ∎

La majorisation est un ordre applicable si on sait que la somme est constante. Mais le fait que la somme reste constante entre les instances n'est pas forcément toujours verifié.

2.1.3.c Un ordre pour l'équité

[Moulin, 1991; Taylor, 2004; Fleurbaey, 2008] proposent une approche axomiatique de la notion d'équité :

Définition 2.33 (Ordre équitable). *Un ordre $\prec$ est dit équitable si et seulement s'il vérifie les propriétés suivantes :*

– *la propriété d'impartialité : toute répartition doit être invariante par permutation des individus. Ainsi des vecteurs égaux à une permutation près, sont considérés équivalents :*

$$\exists \sigma, V_i = V'_{\sigma(i)} \Rightarrow V \equiv V'$$

– *la propriété d'équilibre : diminuer la part d'un utilisateur ne peut pas se compenser par l'augmentation (et encore moins par une diminution) de la part d'un autre utilisateur qui a déjà plus que le premier :*

$$\forall V', \forall i \neq i_1, i \neq i_2 V'_i = V_i, V_{i_1} \leq V_{i_2}, V'_{i_1} \leq V'_{i_2}, V'_{i_1} < V_{i_1} \Rightarrow V' \prec V$$

Le premier axiome d'impartialité s'appuie sur un principe de non corruption : l'ordre ne doit pas changer si on échange les identités. Cela signifie qu'un arbitre ne doit pas décider de la répartition des participants en fonction de leurs identités. Cette qualité est nécessaire pour qu'un arbitrage s'effectue sans suspicion d'impartialité et pour que les utilisateurs reconnaissent au médiateur le droit d'arbitrer un partage sans parti pris.

Le second axiome stipule qu'il n'est pas possible d'améliorer la part reçue par un utilisateur en dégradant la part d'un autre utilisateur qui a déjà moins que le premier. Rappelons que nous utilisons un principe de comparaison, ce qui nous permet de comparer directement les parts reçues par les utilisateurs. A l'aide de ce principe de comparaison on comprend qu'on ne puisse pas justifier ce déséquilibre. Notons que ce principe de comparaison peut comporter des pondérations, en effet tout dépend de ce que l'on cherche à rendre équitable.

La philosophie utilitariste justifie toute dégradation a de la condition d'un utilisateur quelconque si d'autres utilisateurs ont un bien-être qui augmente de plus de a au total. Or, selon nous, ce mécanisme de compensation n'est pas acceptable pour des questions d'équité. Ainsi, le second axiome est raisonnable en ce qu'il déclare ce compromis injustifiable et que toute diminution de la part d'un utilisateur ne peut pas se contrebalancer en augmentant la part de ceux qui ont obtenu plus. Il a été en outre utilisé dans le contexte du calcul distribué [Saule et Trystram, 2009], dans l'analyse multi-critères [Ehrgott, 1995] ou dans la recherche d'allocations équitables [Bouveret et Lemaître, 2006]. Dans l'article de [Moulin, 1991; Maniquet, 2002; Kostreva et al., 2004] est démontré le résultat suivant :

Proposition 2.5 (Leximin)**.** *L'ordre Leximin est le seul ordre qui vérifie cet ensemble d'axiomes.*

On en déduit que pour tout ordre numérique (utilité), il existe toujours des situations où le second axiome n'est pas vérifié et où selon cette philosophie il sera plus "équitable" de diminuer quelqu'un qui a moins pour augmenter la part d'autres qui ont plus. Cela ne nous paraît pas satisfaisant, compte tenu du principe de comparaison.

Notons que la seconde propriété implique que si X est dominé par Y ($X \prec_c Y$) alors $X \prec Y$, car il suffit d'augmenter chaque coordonnée les unes à la suite des autres. Ainsi on a :

Théorème 2.8. *$\prec_c$ est un sous-ordre de $\prec_{Lex}$.*

Démonstration. Soit $V \prec_c V'$ alors $\vec{V} \prec_c \vec{V'}$. Comme $\forall i, \vec{V}_i \leq \vec{V'}_i$ on a $\vec{V} \prec_{Dict} \vec{V'}$ et $V \prec_{Lex} V'$. ■

Ainsi une solution maximale pour $\prec_{Lex}$ est Pareto optimale.

D'autres axiomes existent dont le but est de caractériser la notion d'équité :

Définition 2.34 (Principe de transfert de Pigou-Dalton)**.** *Un ordre vérifie le principe de transfert de Pigou-Dalton si et seulement si*

- $\prec_c$ *est un sous-ordre de* $\prec_D$*,*
- *c'est un ordre impartial*
- *et s'il vérifie la propriété de transfert suivante (transférer* ϵ *de* i_2 *à* i_1 *qui a moins augmente l'équité) :*

$$\forall V, \forall i \neq i_1, i \neq i_2, \forall 0 \leq \epsilon \leq \frac{V'_{i_2} - V'_{i_1}}{2}, V'_i = V_i, V'_{i_1} \leq V'_{i_2}, V_{i_1} = V'_{i_1} + \epsilon, V_{i_2} = V'_{i_2} - \epsilon \Rightarrow V' \prec V$$

L'ordre $\prec_D$ *est obtenu par fermeture transitive de ces deux propriétés ou dit autrement, c'est le plus petit ordre qui vérifie toutes ces propriétés.*

L'ordre $\prec_D$ est un ordre partiel. En effet, $V = [0, 10]$ et $V' = [3, 6]$ sont incomparables, car il ne peut exister aucune suite d'augmentations et de transferts qui transforme un des vecteurs en l'autre. Comme tout transfert et toute augmentation sont croissants pour $\prec_{Lex}$ et que $V \prec_{Lex,\neq} V'$, on ne peut pas atteindre V à partir de V'. On ne peut pas non plus atteindre V' à partir de V car ces transformations sont croissantes pour la somme des composantes. Ainsi $V = [0, 10]$ et $V' = [3, 6]$ sont bien incomparables pour $\prec_D$.

Le principe de transfert de Pigou-Dalton se justifie dans le cas où il est possible d'échanger des ressources où la quantité totale demeure constante. Cependant ce n'est pas généralement le cas.

Proposition 2.6. *Soit un vecteur* V *non dominé pour le problème bi-critère* $[\prec_\sum, \prec_{Lex}]$*, alors* V *est maximal pour* $\prec_D$*.*

Démonstration. Supposons qu'il existe un vecteur V' tel que $V \prec_D V'$, c'est à dire tel qu'il existe une suite d'augmentations et de transferts qui amène de V à V'. Or chaque augmentation et chaque transfert est croissant pour $\prec_{Lex}$. Ainsi $V \prec_{Lex} V'$. De même chaque augmentation et chaque transfert augmente la somme des composantes du vecteur initial, ainsi $\sum_i V_i \leq \sum_i V'_i$. Donc V serait dominé pour le problème bi-critère $[\prec_\sum, \prec_{Lex}]$, ce qui n'est pas par hypothèse. ∎

Ainsi $\prec_D$ peut être vu comme un mélange entre la volonté d'être équitable et la volonté de maximiser la somme (ou la moyenne).

En effet, dans certains problèmes il peut être nécessaire de considérer un critère supplémentaire au critère du Leximin. Prenons un exemple, soient deux utilisateurs (A et B) et deux machines identiques, les tâches sont disponibles à la date 0. Chaque utilisateur possède deux tâches ; l'utilisateur A possède deux tâches op_1^A, op_2^A de durée 4 chacune et l'utilisateur B deux tâches op_1^B, op_2^B de durée respective 3 et 5.

Considérons la solution 1 qui consiste à réserver une machine par utilisateur, ainsi chaque utilisateur a un Makespan de 8. Soit la solution 2 qui consiste à exécuter sur chaque machine les tâches de l'utilisateur A puis celles de l'utilisateur B : sur la première machine on exécute op_1^B suivie de op_1^A et sur la seconde machine on exécute op_2^B suivie de op_2^A. Le Makespan de l'utilisateur A est alors de $C^A_{\max} = 9$ et pour l'utilisateur B de $C^B_{\max} = 5$.

La première solution est certes la plus équitable au sens strict, mais pas forcément acceptable. En effet, il suffit que l'utilisateur A accepte de retarder ses tâches (son Makespan passe ainsi de 8 à 9) pour que la seconde solution soit préférable.

Ainsi, tout dépend du critère que l'on cherche à optimiser. Tout d'abord, il faut choisir avec précaution le principe d'évaluation qui permet de comparer le niveau de service rendu entre les

utilisateurs et de décider ce qu'on veut rendre équitable. De même, selon la nature des problèmes, il est possible de rajouter d'autres critères, comme la somme, et considérer un problème bi-critère Leximin et Somme (voir la remarque sur le principe de transfert de Pigou-Dalton). On peut ajouter des contraintes au problème, ou considérer comme indiscernables de faibles différences sur le critère d'un utilisateur (deux vecteurs sont comparables si chaque composante diffère au moins d'une valeur minimale) et appliquer ensuite le Leximin. Cela revient à prendre en compte un seuil d'erreur possible sur les critères.

[Chiu, 1999] donne plusieurs exemples qui montrent que la clef dans les problèmes de répartition est de définir convenablement un critère de comparaison, pour chaque choix de critère résulte une solution équitable relative à celui-ci.

Exemple

Considérons un problème de choix d'emplacement pour une implémentation d'usine [Ogryczak, 1998] , les distances aux 3 clients principaux sont indiquées en km dans le tableau ci-dessous :

	C_1	C_2	C_3
A	17	86	73
B	34	70	58
C	52	54	46
D	69	41	37
E	54	31	50
F	100	67	88
G	100	102	61
H	100	112	117
I	173	80	99

Tab. 2.2 Distances en km des emplacements pour 3 clients

Nous avons affaire à un problème de minimisation. Ainsi, nous considérons une distance d comme de valeur $-d$ pour la comparaison. On peut alors noter que :

- Les emplacements $\{F, G, H, I\}$ sont dominés.
- Les emplacements C et E minimisent la distance maximale qui est de 54.
- L'emplacement E minimise le Leximin des opposés des distances.

Proposition 2.7. *Soient $A = [a_1, \ldots, a_n]$ et $B = [b_1, \ldots, b_n]$ avec $A \prec_{Lex} B$ alors si on ajoute une même coordonnée à ces deux vecteurs l'ordre est conservé :*

$$[a_1, \ldots, a_n] \prec_{Lex} [b_1, \ldots, b_n] \Leftrightarrow [a_1, \ldots, a_n, x] \prec_{Lex} [b_1, \ldots, b_n, x]$$

Démonstration. On peut supposer les vecteurs A et B triés. Le résultat est vrai si les deux vecteurs sont égaux à une permutation près. Si les vecteurs ne sont pas égaux, notons i_1 le premier indice où $a_{i_1} \neq b_{i_1}$ avec $\forall i < i_1 a_{i_1} = b_{i_1}$.

Si $x \leq a_{i_1}$ alors lorsqu'on trie $[a_1, \ldots, a_n, x]$ et $[b_1, \ldots, b_n, x]$ le nouvel indice de x s'insère entre 1 et i_1 et l'ordre Leximin est preservé. Si $x \geq b_{i_1}$, de même x s'insère aprés i_1 et l'ordre Leximin entre les deux vecteurs ne change pas.

Si $a_{i_1} < x < b_{i_1}$, alors la position de x dans le vecteur trié pour $[a_1, \ldots, a_n, x]$ est aprés a_{i_1} donc l'ordre des i_1 premiers éléments ne change pas. La position pour x dans $[b_1, \ldots, b_n, x]$ trié est avant celle de b_{i_1}.

Comme les vecteurs A et B sont égaux jusqu'à i_1 et que $a_{i_1} < x$ si on note i_2 la position de x dans $[b_1, \ldots, b_n, x]$ alors $a_{i_2} < x$ et l'ordre Leximin entre les deux vecteurs est conservé.

Donc

$$[a_1, \ldots, a_n] \prec_{Lex} [b_1, \ldots, b_n] \Rightarrow [a_1, \ldots, a_n, x] \prec_{Lex} [b_1, \ldots, b_n, x]$$

■

Proposition 2.8. *Soient $\vec{A}, \vec{B}, \vec{C}$ des vecteurs triés avec $\vec{A} \prec_{Lex} \vec{B}$ alors $\vec{A} + \vec{C} \prec_{Lex} \vec{B} + \vec{C}$.*

Démonstration. Comme $\prec_{Lex}$ est équivalent à $\prec_{Dict}$ pour les vecteurs triés et que rajouter un vecteur trié à un autre donne un vecteur trié on obtient le résultat. ■

Attention ce résultat n'est pas vérifié lorsque les vecteurs ne sont pas triés comme le montre l'exemple ci-dessous :

$$[1, 4] \prec_{Lex} [3, 2] \tag{2.1.1}$$
$$[1, 4] + [2, 0] \succ_{Lex} [3, 2] + [2, 0] \tag{2.1.2}$$

Proposition 2.9. *L'ordre Leximin ne passe pas à la limite.*

Démonstration. Soit la suite $\{X^i = [1 + \frac{1}{i}; 1 + \frac{1}{i}]\}$ de points de $\mathbb{R}^2$. On a $\forall i, [1; 2] \prec_{Lex} X^i$ or

$$[1; 1] = \lim_{i \to +\infty} X^i \prec_{Lex} [1; 2]$$

■

Proposition 2.10. *Soit $\{X^i\}, i \in \mathbb{N}$ une suite croissante pour $\prec_{Lex}$ et qui converge vers X. Alors $\forall i, X^i \prec_{Lex} X$*

Démonstration. Nous utilisons le lemme suivant :

Lemme 2.9. *Si X^i converge vers X alors $\vec{X}_i$ converge vers $\vec{X}$.*

Il suffit de démontrer le même résultat mais pour $\prec_{Dict}$.

Supposons qu'il existe un indice a tel que $X \prec_{Dict} X^a$ strictement alors on a $\forall a' \geq a, X \prec_{Dict} X^{a'}$ car la suite est croissante. Soit $a' \geq a$ et $b(a')$ le plus petit indice tel que $X_{b(a')} \neq X^{a'}_{b(a')}$ comme $X \prec_{Dict} X^{a'}$ on a $X_{b(a)} < X^{a'}_{b(a')}$. $\{b(a')\}, a' \geq a$ est une suite d'entiers décroissante car $\{X^i\}$ est croissante pour l'ordre lexicographique. Ainsi $\{b(a')\}, a' \geq a$ doit converger en restant constante après un certain $\bar{a}$, notons b la limite de cette suite. On a alors $\forall a' \geq \bar{a}, \forall b' < b, X_{b'} = X^{a'}_{b'}$ et $X_b < X^{a'}_b$. On conclut que la suite $\{X^i\}, i \in \mathbb{N}$ ne peut pas converger vers X, ce qui est contradictoire.

■

2.1.3.d Paramétrisation de l'ensemble de Pareto avec le Leximin

Définition 2.35 (Translations). *Soit* $\Delta = [\delta_1, \ldots, \delta_p] \in \mathbb{R}^p$*, la translation* T_Δ *est une fonction de* $\mathbb{R}^p \mapsto \mathbb{R}^p$ *qui associe à* $[x_1, \ldots, x_p]$ *le vecteur* $[x_1 + \delta_1, \ldots, x_p + \delta_p]$.

Définition 2.36 (Multiplicateurs). *Un multiplicateur est une fonction de* $\mathbb{R}^p \mapsto \mathbb{R}^p$ *qui associe à* $[x_1, \ldots, x_p]$ *le vecteur* $[\lambda_1 x_1, \ldots, \lambda_p x_p]$*. Le vecteur* $\Lambda = [\lambda_1, \ldots, \lambda_p]$ *est appelé le vecteur coefficient du multiplicateur considéré. On note* $\times_\Lambda$ *le multiplicateur de vecteur coefficient* $\Lambda \in \mathbb{R}^p$.

Théorème 2.10. $\forall \Delta \in \mathbb{R}^N, \Lambda \in \mathbb{R}_*^{+N}, S \in OPT(N, T_\Delta \circ \times_\Lambda, \prec_{Lex}) \Rightarrow S \in OPT(N, id, \prec_c)$

Démonstration. En effet, soit $S \in OPT(N, \times_\Lambda, \prec_{Lex})$. S'il existe S' avec $V(S) \neq V(S')$ et $V(S) \prec_c V(S')$ alors puisque $\forall 1 \leq i \leq N, \lambda_i > 0$, $T_\Delta \circ \times_\Lambda(V(S)) \prec_c T_\Delta \circ \times_\Lambda(V(S'))$.

Comme $S \in OPT(N, T_\Delta \circ \times_\Lambda, \prec_{Lex})$ et que $\prec_{Lex}$ est un ordre total à permutation près, alors $T_\Delta \circ \times_\Lambda(V(S')) \prec_{Lex} T_\Delta \circ \times_\Lambda(V(S))$ et par conséquent $T_\Delta \circ \times_\Lambda(V(S)) = T_\Delta \circ \times_\Lambda(V(S'))$, soit $V(S) = V(S')$, ce qui est contradictoire. ∎

Le résultat suivant montre que réciproquement pour toute solution S de $OPT(N, id, \prec_c)$ il existe un Δ de sorte que S soit solution de $OPT(N, T_\Delta, \prec_{Lex})$:

Théorème 2.11 (Paramétrisation par translations et Leximin). $\forall S \in OPT(N, id, \prec_c), \exists \Delta \in \mathbb{R}^N, S \in OPT(N, T_\Delta, \prec_{Lex})$

Démonstration. Prenons $\Delta = -V(S)$, en effet comme $S \in OPT(N, id, \prec_c)$, pour toutes les autres solutions S', $T_{-V(S)}(S')$ a au moins une coordonnée négative et $T_{-V(S)}(S) = [0, \ldots, 0]$. ∎

Ainsi $OPT(N, T_\Delta, \prec_{Lex})$ paramétrise l'ensemble de Pareto. Ce n'est pas le cas de $OPT(N, X \mapsto \Delta \cdot X, \leq)$ par exemple qui ne parcourt que les solutions non dominées qui sont aussi sur l'enveloppe convexe.

Un résultat similaire s'applique pour les solutions de coordonnées strictement positives :

Théorème 2.12 (Paramétrisation par multiplicateurs et Leximin [Ehrgott, 1995]). $\forall S \in OPT(N, id, \prec_c) \cap \mathbb{R}_*^{+N}, \exists \Lambda \in \mathbb{R}_*^{+N}, S \in OPT(N, \times_\Lambda, \prec_{Lex})$

Démonstration. Il suffit de prendre $\lambda_i = \frac{1}{S_i}$, en effet comme $S \in OPT(N, id, \prec_c)$, pour toutes les autres solutions S', $\times_\Lambda(S')$ a au moins une coordonnée inférieure à 1 et $\times_\Lambda(S) = [1, \ldots, 1]$.

Note : ce résultat peut aussi s'obtenir grâce au précédent en remarquant qu'un point non dominé reste non dominé si on applique une fonction croissante à toutes les coordonnées et en prenant le logarithme. ∎

Ainsi, grâce au Leximin les vecteurs de translation et les multiplicateurs peuvent paramétrer l'ensemble des solutions Pareto optimales. On peut donc faire la correspondance entre l'ensemble de Pareto et une hyper-surface.

Notons qu'il est possible d'imposer que le coefficient du multiplicateur $\times_\Lambda$ soit tel que $\|\Lambda\| = 1$ ou que $\sum_{i=1}^N \lambda_i = 1$ sans changer le résultat, et avoir une application qui associe à tout Λ avec ces contraintes à un point de l'ensemble de Pareto. Cette correspondance permet par exemple de comparer les Λ et de définir des voisinages au lieu de comparer les points de l'ensemble de Pareto.

Proposition 2.11. $S \in OPT(N, T_\Delta, \prec_{Lex}) \Leftrightarrow \forall x \in \mathbb{R}, S \in OPT(N, T_{\Delta + [x, x, \ldots]}, \prec_{Lex})$

Démonstration. La réciproque est vraie en prenant $x = 0$.

Montrons que $A \prec_{Lex} B \Rightarrow A + [x, x, \dots] \prec_{Lex} B + [x, x, \dots]$: On a $A \equiv \vec{A} \prec_{Lex} B \equiv \vec{B}$, or comme $A + [\vec{x}, x, \dots] = \vec{A} + [x, x, \dots]$ et de même pour $B + [\vec{x}, x, \dots]$ l'ordre Leximin compare $\vec{A}_i + x$ et $\vec{B}_i + x$, ce qui est équivalent à comparer $\vec{A}_i$ et $\vec{B}_i$. Ainsi l'ordre Leximin est conservé. ∎

Par conséquent, soit $\Theta : \Delta \mapsto OPT(N, T_{-\Delta}, \prec_{Lex})$ alors $\forall \Delta, x, \Theta(\Delta) = \Theta(\Delta + [x, x, \dots])$, ainsi Δ peut être pris dans tout hyperplan de $[x, x, \dots]$. En particulier $\Theta([0, 0, \dots])$ est $OPT(N, id, \prec_{Lex})$ et si $S \in OPT(N, id, \prec_c)$ alors $S \in \Theta(S)$.

Définition 2.37. *Soit S une solution non dominée, notons E_S l'ensemble des Δ tels que $\Theta(\Delta) = \Theta(S)$.*

Proposition 2.12. *E_S ne contient qu'une solution non dominée à permutation des coordonnées près.*

Démonstration. En effet, soit $S \not\equiv S'$ alors par définition $\Theta(S) \neq \Theta(S')$ donc $E_S \neq E_{S'}$. ∎

Soit E_0 l'ensemble des Δ tels que $\Theta(\Delta) = \Theta([0, 0, \dots])$, E contient la solution S^* maximale pour $\prec_{Lex}$ dans l'ensemble des solutions. D'autre part si la solution S^* est remplacée par une solution non dominée S' qui se trouve dans E_0 alors S' est la nouvelle solution maximale pour $\prec_{Lex}$ dans l'ensemble des solutions. Ainsi E_0 définit une "zone de stabilité" pour S^*.

Observons ainsi que le Leximin parcourt tous les points de l'ensemble de Pareto pourvu que des coefficients soient appliqués aux critères. Chaque point de l'ensemble de Pareto peut donc être considéré comme équitable lorsqu'on lui applique des coefficients appropriés. Cela démontre l'importance du choix du principe de comparaison. D'autre part si on cherche la solution qui se rapproche le mieux d'un vecteur de coefficients de proportionnalité $\forall i, w_i > 0, [\dots, w_i, \dots]$ tout en étant efficace, il suffit de maximiser le Leximin avec des coefficients $[\dots, \frac{x_i}{w_i}, \dots]$. En effet, cette solution a tendance à se rapprocher proportionnellement au vecteur des coefficients w_i sans les inconvénients cités plus haut (avec la mesure de Jain par exemple).

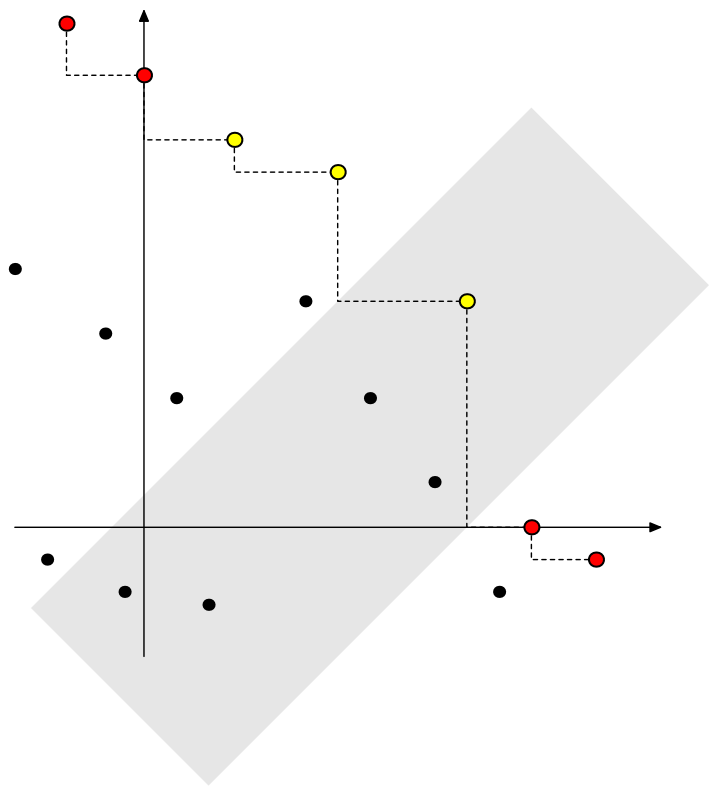


Fig. 2.1.2 Ensemble de Pareto en rouge et jaune, l'ensemble des points jaunes est l'ensemble des points non dominés paramétrables par un multiplicateur. La bande grise est l'ensemble des Δ tels que $\Theta(\Delta) = \Theta([0,0])$ qui contient le point maximal pour $\prec_{Lex}$ dans cet ensemble.

2.1.4 Equité dans le cas stochastique

2.1.4.a Approches classiques

Lorsque le processus de prise de décision doit se répéter parmi un ensemble de stratégies possibles avec des gains additifs et où les probabilités d'occurrence des états sont connues, une règle de décision possible est de maximiser l'espérance E du revenu. Ainsi, la moyenne des revenus converge vers E, néanmoins il est possible que cette convergence manque de régularité pour certaines répartitions de probabilités, ainsi on pourra considérer en plus de l'espérance l'écart-type lorsque la régularité est un critère important.

Cette règle de décision conduit à certains paradoxes, comme celui de Saint-Petersbourg [Samuelson, 1977] où l'espérance du gain est non borné malgré de faibles probabilités de gain. De façon générale, lorsque les probabilités de gain sont très faibles, le coût de participation non négligeable, et que le gain peut être très élevé, maximiser l'espérance du gain aboutit à des comportements "invraisemblables" [Allais, 1953; Ellsberg, 1961].

Rappelons ici que le résultat de l'utilité en espérance ne s'applique logiquement qu'en cas de décisions répétées indéfiniment avec des gains additifs. Dans les autres situations cette méthode de décision n'a pas de fondements solides à être employée.

2.1.4.b Maximiser le Leximin en espérance

Nous nous intéressons maintenant à maximiser le Leximin en espérance lorsque les décisions sont répétées et que les gains sont additifs. Nous disposons d'un ensemble de solutions. Chaque solution est mesurée par un vecteur d'évaluation. Le problème consiste à attribuer une probabilité à l'ensemble des solutions de sorte que l'espérance du vecteur d'évaluation maximise le Leximin.

L'ensemble des valeurs que prend l'espérance du vecteur d'évaluation est donc la fermeture convexe de l'ensemble des vecteurs d'évaluation.

Lemme 2.13. *Un point qui maximise le Leximin sur un convexe $\mathcal{C}$ est sur la frontière de $\mathcal{C}$.*

Démonstration. Montrons qu'un point intérieur à ce convexe ne maximise pas le Leximin : en effet, il existe autour de ce point une boule contenue dans le convexe. Il suffit d'augmenter la coordonnée de valeur minimale de ce point pour obtenir un autre point qui augmente le Leximin. ∎

On en conclut qu'un maximum ne peut se situer que sur la frontière de l'enveloppe convexe des vecteurs d'évaluation. Remarquons qu'un maximum pour le Leximin n'est pas forcément sur un sommet du convexe comme sur l'exemple de la figure 2.1.3.

Lemme 2.14. *Il n'existe qu'un maximum pour le Leximin sur un convexe.*

Démonstration. En effet, supposons qu'il en existe au moins deux distincts notés X et Y. Ces maximums sont équivalents à une permutation près. Or soit Z la moyenne de ces deux points, Z est alors dans le convexe. En effet comme l'ordre des coordonnées est indifférent nous pouvons supposer que X est trié. Prenons la première coordonnée i de X qui est distincte de Y, comme X est trié et que Y est équivalent à X à une permutation près on a $X_i < Y_i$. Ainsi $X \equiv Y \prec_{Lex} Z$, et Z est meilleur pour le Leximin, ce qui est une contradiction. ∎

Ce maximum se situe donc sur une des faces du convexe et peut s'exprimer comme un barycentre des sommets de cette face. Les coefficients de ce barycentre donnent alors les probabilités à attribuer à ces sommets.

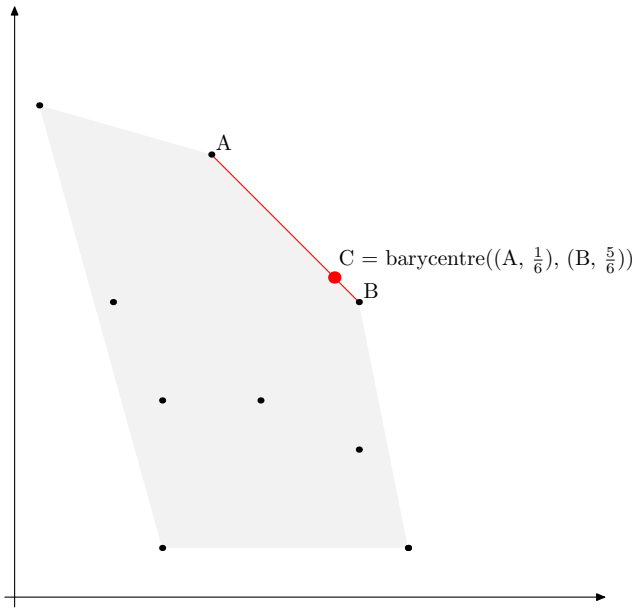


Fig. 2.1.3 Ensemble des vecteurs d'évaluation et son enveloppe convexe. Le point C maximise le Leximin sur ce convexe et ne se trouve pas sur un sommet. C est un barycentre des sommets de la face qui le contient, ici A et B.

Est-ce que les sommets qui participent au barycentre contiennent toujours le point le plus équitable des solutions ? Non car les sommets sont sur la frontière de l'enveloppe convexe or le point le plus équitable peut ne pas s'y trouver, par exemple prenons les 4 points de la figure suivante 2.1.4, le point rouge est celui qui maximise le Leximin et il n'est pas sur l'enveloppe convexe.

Comment trouver ce maximum, les sommets de la face qui le contiennent et les probabilités à attribuer à ces sommets ? Nous avons déjà vu que les probabilités correspondent aux coefficients barycentriques attribués aux sommets. Il suffit alors de trouver les sommets et ensuite de calculer les coefficients de façon à maximiser le Leximin.

Théorème 2.15. *Soit la face qui contient le maximum pour le Leximin des points du convexe, alors cette face contient les sommets qui maximisent le Leximin sur l'enveloppe convexe.*

Démonstration. Prenons un sommet S du convexe, on sait que le maximum fait parti des points du convexe qui sont au moins aussi bons que S. Ainsi il est possible de considérer seulement l'ensemble des points qui appartiennent au convexe et qui sont meilleurs que S. Si on poursuit ce processus avec les sommets restants on aboutit à un sous-ensemble du convexe qui contient un ensemble de sommets équivalents pour le Leximin. Cet ensemble est non vide car à chaque étape on garde le sommet considéré, il contient les sommets qui maximisent le Leximin parmi les sommets du convexe. ■

Il faut toutefois remarquer que tous les points de la face contenant le maximum pour le Leximin peuvent être tous dominés par d'autres points : c'est le même problème que pour la moyenne, voir la figure 2.1.5. Tous les points de l'ensemble $\{A^1, B^1\}$ sont moins équitables que les points de

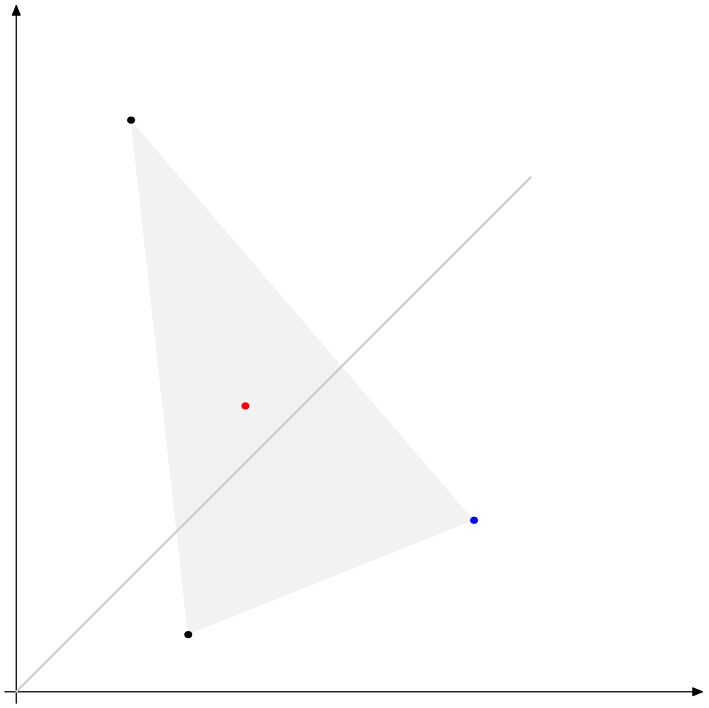


Fig. 2.1.4 Considérons l'ensemble de ces quatres points, le point rouge est celui qui maximise le Leximin et ne se situe pas sur la frontière de l'enveloppe convexe. Par contre, le point qui maximise le leximin sur l'enveloppe convexe est sur une face qui contient le maximum pour le leximin parmi les sommets de la frontière (ici le point bleu).

l'ensemble $\{A^2, B^2\}$, on peut dire aussi que le vecteur $[A^1, B^1]$ est dominé par le vecteur $[A^2, B^2]$ pour le Leximin sur les coordonnées. Pourtant, la moyenne M^1 de l'ensemble $\{A^1, B^1\}$ est plus équitable que la moyenne M^2 de l'ensemble $\{A^2, B^2\}$. Ainsi, un critère d'agrégation comme la moyenne n'est pas Pareto efficace pour le Leximin : maximiser la moyenne d'un vecteur dont les composantes sont des vecteurs ne conduit pas forcément à un vecteur non dominé pour le Leximin sur les coordonnées. On en conclut qu'être équitable sur une moyenne même pondérée n'est pas un critère valide si le problème n'est pas répété ou si les gains ne sont pas additifs.

Dans le cas probabiliste prendre la moyenne pondérée des allocations par les probabilités a-t-elle un sens ? Non, car si les vecteurs ne sont pas triés avant d'en faire la somme, on a l'ordre de la moyenne ($\ll^{\Sigma}$) et cet ordre ne respecte pas le Leximin comme le montre l'exemple de la figure 2.1.5. Ainsi on ne fait pas de moyennes directement sur les vecteurs. En revanche, faire la moyenne sur les vecteurs triés respecte bien le Leximin, comme nous le verrons dans la section suivante.

Ainsi l'approche classique qui consiste à considérer l'enveloppe convexe par pondération stochastique n'est pas une approche valide pour l'ordre Leximin pour un problème de décision non répété. Dans la section suivante nous rechercherons à conserver la compatibilité avec le Leximin.

2.1.4.c Critique des méthodes d'optimisation basées sur l'espérance

D'Alembert [D'Alembert, 1764] a remarqué que

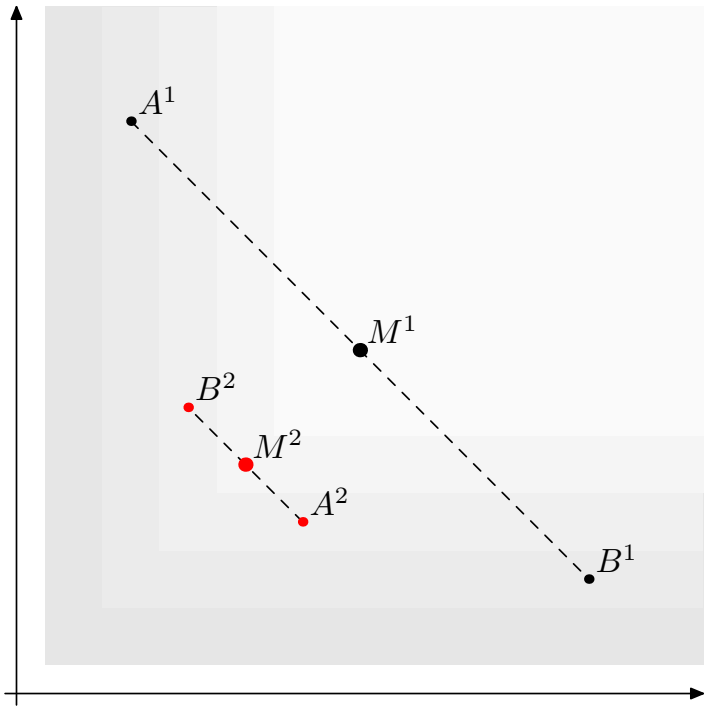


Fig. 2.1.5 .

> « La règle ordinaire de l'analyse des jeux de hasard est celle-ci : multipliez les gains ou la perte que chaque événement doit produire par la probabilité qu'il y a que cet événement doit arriver ajoutez ensemble tous ces produits [...] et vous aurez l'espérance du joueur, ou ce qui revient au même ce que le joueur devrait donner avant le jeu pour commencer à jouer but-à-but *(i.e. également, sans aucun avantage de part ni d'autre)*. Aucune analyse que je sache n'a jusqu'ici révoqué cette règle en doute et tous s'y sont conformés dans les calculs qu'ils ont faits des différentes probabilités. Il se trouve néanmoins des cas où elle parait être mis en défaut, et qui vont faire la matière de quelques réflexions. »
>
> [D'Alembert, 1761]

Il poursuit en rejetant le critère de la maximisation de l'espérance (ou même d'une utilité) comme méthode de décision. De même John Keynes [Keynes, 1921; Henry, 2005] a rejeté la méthode de maximisation de l'espérance du gain en remarquant que la prise de risque engendrée peut elle-même invalider une décision (ce qui justifie les approches basées sur la robustesse).

Tout comme dans le processus de décision multicritères le critère de la moyenne n'est donc pas pleinement satisfaisant pour exprimer la diversité des choix. Etant donné que certains choix sont par nature incomparables entre eux, on peut tirer un parallèle avec le processus de décision avec prise en compte des risques.

Ainsi dans le cas où l'expérience n'est pas répétée on préferera utiliser la notion de *dominance stochastique*, similaire à la notion de Majorisation dans le cas discret :

Définition 2.38 (Dominance stochastique [Fishburn, 1964; Hadar et Russell, 1969; Bawa, 1975])**.** *Une variable aléatoire X domine stochastiquement une variable aléatoire Y ($X \prec_{Stoch} Y$) si*

et seulement si on a

$$\forall a, Pr[X > a] \geq Pr[Y > a]$$

Propriété 2.1 ([Whitmore, 1989])**.**

$$X \prec_{Stoch} Y \Rightarrow E[X] \geq E[Y]$$

De plus, pour toute fonction f croissante on a :

$$X \prec_{Stoch} Y \Rightarrow E[f(X)] \geq E[f(Y)]$$

Il est clair qu'une telle notion de dominance est un ordre partiel. On retrouve les mêmes préoccupations tant pour le choix d'une décision particulière que pour l'optimisation multi-critères : certaines décisions ne sont tout simplement pas comparables.

2.2 Comparaison d'ensembles ordonnés valués

Nous parlons ici du problème de la comparaison de deux sous-ensembles d'un ensemble ordonné, où on associe à chaque élément une pondération. En particulier, cela s'applique à la comparaison de distribution de probabilités. Prenons un exemple de problèmes où on est amené à comparer des distributions de probabilité sur des sous-ensembles de $\mathbb{R}^N$.

Exemple de comparaison de distributions de probabilités sur des ensembles ordonnés

Un administrateur doit partitionner l'ensemble d'un volume de stockage entre plusieurs groupes d'utilisateurs. Ainsi s'il attribue à un groupe i une partition des ressources de capacité C^i le groupe pourra utiliser cette ressource jusqu'à C^i (et devra éventuellement utiliser d'autres ressources moins performantes au delà). Si $[V^1, \ldots, V^N]$ représente le vecteur des demandes en stockage, le vecteur d'allocation sera alors $[\min(V^1, C^1), \ldots, \min(V^N, C^N)]$. Supposons que la demande des groupes est incertaine et qu'on connaisse la distribution de probabilités des demandes des groupes, i.e. on associe une probabilité $p([V^1, \ldots, V^N])$ à la demande $[V^1, \ldots, V^N]$. On obtient alors une distribution de probabilités pour les allocations qui est $p([V^1, \ldots, V^N])$ pour l'allocation $[\min(V^1, C^1), \ldots, \min(V^N, C^N)]$. Cette distribution ne dépend que de la partition $[C^1, \ldots, C^N]$ choisie *a priori*. On est ainsi amené à devoir comparer des distributions de probabilités sur des sous-ensembles de $\mathbb{R}^N$.

2.2.1 Comparaison de politiques lorsque les probabilités d'occurrence sont connues

2.2.1.a Étude d'un cas simple où il ne reste qu'une machine pour deux utilisateurs

On suppose que l'allocation des machines est $(a, b), a \leq b$: le premier utilisateur dispose de a machines et le second de b. Soit p et q les probabilités que le premier (respectivement. le second) utilisateur demande au moins une machine de plus. On considère la demande actuelle.

Si aucun des deux utilisateurs n'a et n'aura de demandes supérieures à son allocation actuelle alors l'allocation restera comme telle quelle (a, b).

	$1-p$	p
$1-q$	(a,b)	$(a+1,b)$
q	(a,b) $(a,b+1)$	$(a+1,b)$ $(a,b+1)$

Tab. 2.3 Cas possibles

Si chaque utilisateur a une demande plus grande que son allocation actuelle alors cela ne sert à rien de ne pas allouer la dernière machine à un des deux utilisateurs, en particulier dans ce cas à celui qui a reçu le moins.

Si c'est le premier utilisateur qui a fait une demande supplémentaire le premier alors comme $(a,b) \prec (a,b+1) \prec (a+1,b)$, $(a+1,b)$ est le maximum possible donc il faut allouer une machine au premier utilisateur. On ne sait pas si le second utilisateur va demander une machine supplémentaire mais on est certain d'être au maximum.

Si c'est le second utilisateur qui fait une demande supplémentaire le premier, alors on est dans l'incertitude car on ne sait pas si le premier utilisateur va demander une machine ou pas. Dans ce cas si on alloue une machine au second utilisateur on a l'allocation $(a,b+1)$ avec une probabilité de 1.0. Par contre si on ne lui alloue pas on a (a,b) avec une probabilité de $1-p$ et $(a+1,b)$ avec une probabilité de p. Il reste donc à comparer les distributions de probabilités dans ces deux cas : nous avons affaire à un dilemme : Décider d'une allocation à un instant donné a une incidence sur les répartitions futures possibles : De façon générale si on donne une machine il y a la possibilité de leser plus tard un autre utilisateur par manque de machines disponibles et si on ne donne pas il y a la possibilité de gaspillage. Il y a donc une notion de *risque* dans l'allocation sous incertitudes. La décision dépendra de la mesure de comparaison choisie.

De même pour le cas où les demandes évoluent, dans ce cas on associe à chaque scénario une probabilité, par conséquent chaque politique est valuée. On doit choisir entre plusieurs ensembles de probabilités sur un ensemble E muni d'un critère. A la suite d'un événement un élément de E est tiré aléatoirement suivant les probabilités de la distribution choisie.

Comment alors comparer les distributions de probabilités entre elles ?

Définition 2.39. *Soit E un ensemble, on appelle distribution de probabilités sur E toute fonction $p : E \mapsto [0,1]$ telle que $\sum_{x \in E} p(x) = 1$. Notons $\mathbb{P}_E$ l'ensemble de ces distributions de probabilités sur E.*

Définition 2.40 (Support d'une distribution de probabilités)**.** *Le support d'une distribution de probabilités p est l'ensemble des points $x \in E$ tels que $p(x) \neq 0$.*

Nous considérons uniquement les distributions dont le support est un ensemble fini.

Nous nous intéresserons au problème de la comparaison de probabilités sur des ensembles de vecteurs munis d'un ordre $\prec$ (en particulier l'ordre Leximin). Comment comparer ces probabilités ? Soit $\ll$ un ordre sur $\mathbb{P}_E$. Tout d'abord nous introduisons des propriétés souhaitables sur les ordres entre probabilités.

Une des propriétés souhaitables sur les ordres de probabilités est la suivante :

Définition 2.41 (Respect de $\prec$)**.** *Soit $\ll$ un ordre sur $\mathbb{P}_E$. On dit que $\ll$ respecte $\prec$ si et seulement si pour toute loi de probabilité p s'il existe une bijection f croissante pour $\prec$ du*

support de p vers un sous-ensemble E' de E alors la loi de probabilité p' telle que $p'(f(x)) = p(x)$ alors $p \ll p'$.

En particulier si $\ll$ respecte $\prec$ alors si on remplace un point x de E par un point x' avec $x \prec x'$ on a $p \ll p'$. Notons que cela ne modifie pas la valeur des probabilités mais seulement les points.

Définition 2.42 (Transfert). *Soit p une distribution de probabilités et x tel que $p(x) \geq \delta$ et $p(y) + \delta \leq 1$. Soit p' la distribution de probabilités telle que :*

$$\begin{aligned} \forall z, z \neq x, z \neq y, p'(z) &= p(z) \\ p'(x) &= p(x) - \delta \\ p'(y) &= p(y) + \delta \end{aligned} \tag{2.2.1}$$

On dit que p' est obtenue à partir de p par transfert de δ de x vers y. Nous notons $\theta[x, y, \delta]$ la fonction de transfert avec $p' = \theta[x, y, \delta](p)$.

Il est souhaitable d'avoir aussi la propriété suivante :

Définition 2.43 (Extension de $\prec$). *Soit p une probabilité. Si $\ll$ à la propriété d'extension de $\prec$ alors $p \ll p'$ pour tout p' obtenue à partir de p par une suite éventuellement infinie de transferts $\theta[x, y, \delta]$ tels que $x \prec y$.*

Cela correspond au fait d'augmenter la probabilité sur un point x et de la diminuer sur des points y avec $y \prec x$, alors le nouvel ensemble est meilleur.

Définition 2.44 (Plongement de C). *On plonge l'ensemble C des scénarios possibles dans l'ensemble des distributions de probabilités sur C de la façon suivante : on associe à $c \in C$ p^c telle que*

$$p^c(x) = \begin{cases} 0 & \text{si } x \neq c \\ 1 & \text{si } x = c \end{cases}$$

On en déduit que les ordres $\ll$ qui respectent $\prec$ vérifient : $p^{\min_\prec \mathbb{S}_p} \ll p \ll p^{\max_\prec \mathbb{S}_p}$. Ainsi on peut voir $\ll$ comme une extension de $\prec$ sur $\mathbb{P}_E$. L'objectif est de savoir généraliser l'ordre $\prec$ sur tout l'ensemble C et non uniquement sur les distributions p^x.

Théorème 2.16 (Extension $\Rightarrow$ Respect). *Si $\ll$ est une extension de $\prec$ alors $\ll$ respecte $\prec$.*

Démonstration. Soit le transfert de $p(X)$ de X vers $f(X)$ $\theta[X, f(X), p(X)]$. Comme f est croissante on a $p \ll \theta[X, f(X), p(X)](p)$ car $\ll$ est supposée être une extension de $\prec$.

Si on applique cette transformation sur tous les points du support de p (qui est un ensemble fini) on obtient une densité de probabilité q telle que $q(Y) = p(f^{-1}(Y))$ et $q(Y) = 0$ s'il n'existe pas de X tel que $f(X) = Y$. q est bien une densité de probabilité car

$\forall Y, 0 \leq q(Y) = p(f^{-1}(Y)) \leq 1$

et $\sum_Y q(Y) = \sum_X p(X) = 1$

Et par applications successives, si $\ll$ est une extension de $\prec$ alors elle respecte $\prec$.

■

2.2.2 Optimisation stochastique et fonction d'utilité de Von Neumann-Morgenstern [Neumann et Morgenstern, 1944]

Comment comparer deux politiques d'allocation dans le cas où les probabilités des différents scénarios sont connues ? Il est possible de définir une probabilité qu'une politique A soit meilleure qu'une politique B sur l'ensemble des scénarios possibles. Néanmoins cette mesure risque d'être biaisée en faveur des décisions qui maximisent l'objectif pour les scénarios les plus probables.

Soit $\prec$ une relation d'ordre sur l'ensemble P des distributions de probabilités à support fini [2] sur un ensemble d'événements C éventuellement fini composé de n événements. $p(c)$ désigne la probabilité que la distribution p assigne à l'événement c.

Exemple.

Considérons le cas où la demande des utilisateurs est incertaines et peut prendre la valeur $[1,2,6]$ avec la probabilité de 0.6 ou la valeur $[3,1,5]$ avec la probabilité 0.4. Supposons que nous devons partitionner 9 unités de ressources. Soit la partition uniforme $P^1 = [3,3,3]$, elle attribue dans le premier cas $\min_c([3,3,3],[1,2,6]) = [1,2,3]$ et dans le second cas $\min_c([3,3,3],[3,1,5]) = [3,1,3]$. Soit la partition $P^2 = [3,2,4]$ qui attribue dans le premier cas $[1,2,4]$ et dans le second cas $[3,1,4]$. Chaque partition définit alors une distribution de probabilités sur l'ensemble des vecteurs d'allocations possibles.

Définition 2.45 (Pondérations de distributions de probabilités)**.** *On note $\alpha p + (1-\alpha)q$ la distribution de probabilités qui assigne à l'événement c quelconque la probabilité $\alpha p(c) + (1-\alpha)q(c)$.*

John von Neumann et Oskar Morgenstern [Neumann et Morgenstern, 1944] vont poser les bases de la théorie de l'espérance en utilité. Ils proposent l'ensemble d'axiomes suivant :

Axiome 2.1 (VNM 1 : (Relation complète et transitive))**.** *$\prec$ est une relation sur P*
- *complète ($\forall a, b, a \prec b$ ou $b \prec a$ ou $a \equiv b$)*
- *et transitive ($a \prec b$ et $b \prec c \Rightarrow a \prec c$)*

Axiome 2.2 (VNM 2 : (Indépendance))**.** *Pour tout $p, q, r \in P$ et $\alpha \in]0,1]$, si $p \prec q$ alors $\alpha p + (1-\alpha)r \prec \alpha q + (1-\alpha)r$.*

Proposition 2.13. *L'axiome d'indépendance implique que :*
- $\forall \alpha \in]0,1], p \prec q \Rightarrow \alpha p + (1-\alpha)q \prec q$
- $\forall \alpha \in]0,1], p \prec q \Rightarrow p \prec \alpha p + (1-\alpha)q$
- $\forall \alpha, \beta \in]0,1], \alpha \leq \beta, p \prec q \Rightarrow (1-\alpha)p + \alpha q \prec (1-\beta)p + \beta q$

Démonstration. La première inégalité s'obtient en prenant $r = q$ et la seconde avec $r = p$. ∎

D'autre part, on généralise aussi l'ordre $\prec$ sur C : si $x \prec y$, alors $p^x \prec' p^y$.

Définition 2.46 (Compatibilité)**.** *Un ordre $\prec'$ sur les distributions de probabilités est dit* compatible *avec un ordre sur les éléments de C si et seulement si :*
- *$\prec'$ vérifie l'axiome d'indépendance*
- *et $\forall x, y \in C, x \prec y \Rightarrow p^x \prec' p^y$.*

2. i.e. telle que l'ensemble des éléments de probabilité non nulle est fini.

Suite de l'exemple.

Reprenons notre exemple et munissons l'ensemble des vecteurs d'allocations de l'ordre Leximin. Il devient alors clair que la partition P^2 est plus équitable que la partition P^1 car $[1,2,3] \prec_{Lex} [1,2,4]$ et $[3,1,3] \prec_{Lex} [3,1,4]$. Cela se vérifie d'après l'axiome VNM2. Cet axiome est suffisant pour justifier la définition de la dominance stochastique.

Comment caractériser les ordres compatibles avec un ordre donné ?

Axiome 2.3 (VNM 3 : (Principe d'Archimède)). *Si $p \prec q \prec r$ alors il existe $\alpha \in]0,1]$ tel que $\alpha p + (1-\alpha r) \equiv q$.*

Proposition 2.14 ([Neumann et Morgenstern, 1944]). *Les axiomes VNM1, VNM2 et VNM3 impliquent qu'il existe une fonction linéaire $u : P \mapsto \mathbb{R}$ telle que $p \prec q \Leftrightarrow u(p) \geq u(q)$. De plus u est unique, à une transformation affine près. (i.e. l'ordre $\prec_u$ défini par la fonction u est compatible avec $\prec$.)*

Notons que la linéarité de u implique que :

$$\forall p \in P, u(p) = \sum_{c \in C} p(c) u(p^c)$$

Supposons que l'ensemble C est muni d'une relation d'ordre totale mais non représentable par une fonction scalaire (comme l'ordre lexicographique ou le Leximin) alors une conséquence de ce théorème est que nous ne pouvons pas étendre cet ordre en respectant tout les axiomes, sinon la fonction d'utilité u ci-dessus pourrait représenter l'ordre sur C, ce qui impossible par hypothèses.

L'axiome le plus contestable du point de vue de l'équité est le principe d'Archimède, puisque pour le Leximin il postule que toute injustice dans un scénario peut être contrebalancée par plus d'équité dans un autre scénario. Ainsi cet axiome est l'analogue de la philosophie utilitariste mais appliquée aux événements. En effet, considérons ce que l'on obtient lorsque l'axiome VNM3 n'est pas vérifié :

Proposition 2.15 (Universalité de l'ordre lexicographique [Fishburn, 1982; Gottinger, 1982; L. et al., 1989]). *Supposons l'ensemble C de cardinalité finie n. Les axiomes VNM1 et VNM2 sont vérifiés si et seulement s'il existe un entier $K < n$ et des fonctions linéaires $u_1, \ldots, u_K$ sur P telles que pour tout $p, q \in P$:*

$$p \prec q \Leftrightarrow [u_1(p), \ldots, u_K(p)] \prec_{Dict} [u_1(q), \ldots, u_K(q)]$$

De plus, chaque fonction $u_1, \ldots, u_K$ est unique à une pondération près.

Notons $U(p) = [u_1(p), \ldots, u_K(p)]$. Remarquons de même que la linéarité des fonctions $u_1, \ldots, u_K$ donne :

$$\forall p \in P, U(p) = [u_1(p), \ldots, u_K(p)] = \sum_{c \in C} p(c)[u_1(p^c), \ldots, u_K(p^c)] \tag{2.2.2}$$

$$= \sum_{c \in C} p(c) U^c \tag{2.2.3}$$

$$\text{avec } U^c = [u_1(p^c), \ldots, u_K(p^c)] \tag{2.2.4}$$

$$\tag{2.2.5}$$

Comme $p \prec q \Leftrightarrow U(p) \prec_{Dict} U(q)$, $p \prec q \Leftrightarrow \sum_{c \in C} p(c)U^c \prec_{Dict} \sum_{c \in C} q(c)U^c$. Ce résultat très important peut être utilisé lorsque C est un ensemble de vecteurs et que $\prec$ est le Leximin, lorsqu'on doit comparer des distributions de probabilités sur des ensembles de vecteurs avec l'ordre Leximin. Ainsi si on souhaite avoir un ordre sur les politiques qui est compatible avec l'ordre Leximin cela implique que $c^1 \prec_{Lex} c^2 \Rightarrow U^{c^1} \prec_{Lex} U^{c^2}$.

En particulier citons un ordre qui respecte le Leximin sur les distributions de probabilités sur les vecteurs (tel que $A \prec B \Rightarrow p^A \ll p^B$) et l'axiome d'indépendance VNM2. En effet un ordre sur les politiques doit respecter le Leximin sinon on risque de choisir une politique dominée. Si on trie avant de faire la moyenne cet ordre étend $\prec_{Lex}$ car le Leximin est alors équivalent à l'ordre lexicographique dans le cône des vecteurs triés :

Définition 2.47. *Soit $\ll^{\vec{\Sigma}}$ définit par $p \ll^{\vec{\Sigma}} q \Leftrightarrow \sum_X p(X)\vec{X} \prec_{Lex} \sum_X q(X)\vec{X}$*

Suite de l'exemple

Rappelons que P^1 consiste à partionner uniformément ($[3,3,3]$), avec une probabilité de 0.6 on obtient $[1,2,3]$ et $[3,1,3]$ avec une probabilité de 0.4. La politique P^2 est la partition $[3,2,4]$ et obtient dans le premier cas $[1,2,4]$ et $[3,1,4]$ dans le second cas.

Pour P^1 la somme pondérée et triée donne : $0.6[1,2,3] + 0.4[1,3,3] = [1,2.4,3]$ et pour P^2 : $0.6[1,2,4] + 0.4[1,3,4] = [1,2.4,4]$. Ainsi $P^1 \ll^{\vec{\Sigma}} P^2$, ce qui était attendu puisque P^2 est plus équitable quelque soit le scénario.

Définition 2.48. *Soit $\equiv^{\vec{\Sigma}}$ telle que $p \ll^{\vec{\Sigma}} q$ et $q \ll^{\vec{\Sigma}} p \Leftrightarrow p \equiv^{\vec{\Sigma}} q$.*

Théorème 2.17. *$\equiv^{\vec{\Sigma}}$ est une relation d'équivalence.*

Démonstration. En effet $\equiv^{\vec{\Sigma}}$ est réfléxive car $\sum_X p(X)\vec{X} \prec_{Lex} \sum_X p(X)\vec{X}$. De plus $\equiv^{\vec{\Sigma}}$ est symétrique de par sa définition et transitive car $\prec_{Lex}$ l'est. ∎

Théorème 2.18. *$\ll^{\vec{\Sigma}}$ est un ordre total sur l'ensemble des vecteurs muni de la relation d'équivalence $\equiv^{\vec{\Sigma}}$.*

Démonstration. $\ll^{\vec{\Sigma}}$ est réfléxive et anti-symmétrique d'après la définition de $\equiv^{\vec{\Sigma}}$. $\ll^{\vec{\Sigma}}$ est transitive car $\prec_{Lex}$ l'est. ∎

Théorème 2.19. *$\ll^{\vec{\Sigma}}$ est compatible avec le Leximin.*

Démonstration. Montrons que cet ordre respecte le Leximin et vérifie l'axiome d'indépendance. $\ll^{\vec{\Sigma}}$ respecte le Leximin car si $A \prec_{Lex} B$ alors $\sum_X p^A(X)\vec{X} = \vec{A} \equiv A \prec_{Lex} B \equiv \vec{B} = \sum_X p^B(X)$. $\ll^{\vec{\Sigma}}$ vérifie l'axiome d'indépendance car soit $p \ll^{\vec{\Sigma}} q$ on a $\sum_X(\alpha p + (1-\alpha)r)(X)\vec{X} = \alpha(\sum_X p(X)\vec{X}) + (1-\alpha)(\sum_X r(X)\vec{X})$. Or, comme $\sum_X p(X)\vec{X} \prec_{Lex} \sum_X q(X)\vec{X}$ on a $\alpha \sum_X p(X)\vec{X} \prec_{Lex} \alpha \sum_X q(X)\vec{X}$ et puisque $(1-\alpha)(\sum_X r(X)\vec{X})$ est un vecteur trié on obtient que $\alpha p + (1-\alpha)r \ll^{\vec{\Sigma}} \alpha q + (1-\alpha)r$. ∎

Associons à une distribution de probabilité p le vecteur suivant obtenu par adjonctions successives :

$$\oplus : p \mapsto \oplus(p) = \bigoplus_x p(x)x$$

Définition 2.49. *Soit l'ordre $\prec_{\oplus}$ défini par $p \prec_{\oplus} q \Leftrightarrow \oplus(p) \prec_{Lex} \oplus(q)$*

Proposition 2.16. *$\prec_{\oplus}$ étend le Leximin mais n'est pas compatible avec le Leximin.*

Démonstration. $\prec_{\oplus}$ étend l'ordre Leximin car $\oplus(p^a) = a$, ainsi $a \prec_{Lex} b \Rightarrow \oplus(p^a) \prec_{Lex} \oplus(p^b) \Rightarrow p^a \prec_{\oplus} p^b$. Si $p \prec_{\oplus} q$ alors $\alpha p \prec_{\oplus} \alpha q$ puisque c'est le cas pour le Leximin. Cependant comme le Leximin n'est pas préservé par l'addition, $\prec_{\oplus}$ ne vérifie pas l'axiome d'indépendance. ■

Nous venons de décrire dans ce chapitre le cadre général de l'optimisation multi-utilisateurs. Nous allons maintenant nous intéresser plus particulièrement au problème d'allocation de ressources dont les demandes peuvent comporter des incertitudes.

Chapitre 3

Équité et allocation de ressources

> Le prince ne doit pas craindre de n'avoir pas une population nombreuse, mais de ne pas avoir une juste répartition des biens.
>
> Confucius

> Par bonne distribution, il faut entendre non une distribution égale, mais une distribution équitable. La première égalité, c'est l'équité.
>
> Victor Hugo

Dans ce chapitre, il sera question du problème de l'allocation de ressources avec pour objectif d'être équitable envers les utilisateurs. Des ressources doivent être allouées de façon permanentes à des utilisateurs. Ces ressources sont considérées comme générales et peuvent être diverses comme des espaces mémoire, des partitions de disque, de la bande passante réseau, des bandes de fréquences [Ahmed et al., 2009], des orbites satellitaires [Bouveret et Lemaître, 2006], voire l'approvisionnement en eau [Wang et al., 2004], etc.

La question de l'équité se divise en deux groupes distincts :

- Tout d'abord celui de la *répartition* où plusieurs utilisateurs font des demandes à un gestionnaire de ressources qui décide de la répartition à donner à chacun. La fonction du gestionnaire est de répartir les ressources entre utilisateurs de la façon la plus équitable possible.
- Un autre groupe de problème est celui du *partage* où plusieurs fournisseurs indépendants approvisionnent en ressources plusieurs clients. De même on souhaite que la part donnée par les fournisseurs ainsi que la part reçue par les clients soit répartie de la façon la plus équitable possible.

Nous nous interesserons dans ce chapitre au problème de répartition dans lequel la gestion des ressources est sous la responsabilité d'un gestionnaire.

Les besoins des utilisateurs ne peuvent pas être totalement prévus. Toutefois un planning prévisionnel de l'utilisation des ressources où différents scénarios sont envisagés, est établi. Ce planning renseigne sur l'évolution des demandes.

Lorsqu'un groupe d'utilisateurs se partage une ressource commune, ce type de problème survient . Lorsque ces utilisateurs ont les mêmes droits d'accès aux ressources, se pose le problème du partage équitable de ces ressources selon les différents besoins des utilisateurs. La problèmatique d'un gestionnaire ou d'un arbitre équitable est présente dans [Myerson, 1991; Young, 1994].

[Knight, n.d.] établit la distinction entre risque et incertitude. En effet, nous ne disposons souvent pas d'assez d'informations (statistiques ou autres) pour évaluer avec précision la distribution

de probabilités des événements possibles. Dans le cas où on ne peut pas attribuer de probabilités aux différents scénarios on parle de décisions sous incertitudes.

Nous proposons une approche basée sur le fait de considérer l'aboutissement de chaque scénario comme un *critère* du problème. Cette approche est analysée en détail dans [Hites et al., 2006]. Cela permet de réduire un problème sous incertitudes à l'étude d'un problème multi-objectifs avec un objectif par scénario résultants.

Nous avons considéré le problème sous l'angle d'un modèle multi-critères. Chaque critère correspond à un scénario [Hites et al., 2006]. Une solution est une politique d'allocation qui attribue les ressources compte-tenu de l'incertitude sur les demandes.

Nous présonterons un algorithme qui donne une liste de toutes les politiques non dominées selon toutes les évolutions possibles des demandes des utilisateurs.

Une bonne gestion des ressources permet d'éviter que celles-ce soient mal réparties ou dilapidées pendant que d'autres manquent du minimum. Étant donné que les ressources sont limitées, la gestion de ces ressources est laissée à une autorité de régulation, en principe neutre et équitable. Ainsi un gestionnaire aura pour devoir de réguler la consommation des ressources partagées afin que chaque utilisateur puisse obtenir une part équitable du système. De plus en cas de conflits, le gestionnaire doit non seulement prendre des décisions mais aussi être capable de justifier ses choix.

Un mécanisme d'allocation de ressources doit être capable de s'adapter à des demandes spécifiques. En effet, le site a un contrôle de ses ressources et peut les attribuer comme il l'entend entre différents groupes ou organisations. En revanche, la portion partagée entre plusieurs groupes ayant les mêmes droits d'accès à la ressource se doit d'être répartie de la façon la plus équitable possible.

La façon dont les ressources sont utilisées n'est pas du ressort du gestionnaire de ressources : Par exemple après allocation d'un ensemble de machines à un utilisateur le gestionnaire n'est pas responsable du placement des tâches effectuées par l'utilisateur sur ces machines.

Exemple : Évolution planifiée des demandes

Les demandes des différents utilisateurs sont planifiées, en sorte qu'aux différents points de décision le gestionnaire pourra prévoir et établir les allocations. Par exemple dans le cadre du projet LHC, une planification de l'utilisation des ressources sur 3 années est établie (voir figure 3.1), celle-ce tient compte des demandes en ressources de stockage et de calcul sur l'ensemble des différents sites participants. Chaque site s'engage alors en signant un "Memorandum of Understanding" qui stipule les quantités de stockage et de calcul que le site promet de fournir aux différents groupes. Un défaut de ce mécanisme est que le site est passif devant la soumission des tâches des utilisateurs et ne peut donc pas en toute honnêteté garantir une part en calcul, si les tâches ne parviennent pas au site sachant qu'en pratique le nombre de tâches est très fluctuant et que les tâches de longue durée peuvent monopoliser un site plusieurs jours - voir aussi les problèmes associés aux WMS présentés dans l'introduction.

CMS	2010	2011	2012	ATLAS	2010	2011	2012
CERN CPU (kHEPSpec 2006)	96.6	106.1	106.1	CERN CPU (kHEPSpec 2006)	67	75	28
CERN Disque (PetaBytes)	4.1	4.5	4.5	CERN disque (PetaBytes)	3.9	7	7.5
CERN Bande (PetaBytes)	14.6	21.6	24.6	CERN Bande (PetaBytes)	9	12.2	12.5
T1 CPU (kHEPSpec 2006)	100.5	150.7	150.7	T1 CPU (kHEPSpec 2006)	192	226	223
T1 Disque (PetaBytes)	13.4	19.5	25.1	T1 Disque (PetaBytes)	21.9	24.8	27
T1 Bande (PetaBytes)	23.3	52.4	64.7	T1 Bande (PetaBytes)	14.2	30.1	39
T2 CPU (kHEPSpec 2006)	195	319.5	319.5	T2 CPU (kHEPSpec 2006)	240	278	295
T2 Disque (PetaBytes)	9.2	19.9	23	T2 Disque (PetaBytes)	20.9	37.6	44
ALICE	2010	2011	2012	LHCb	2010	2011	2012
CERN CPU (kHEPSpec 2006)	46.8	62	65.7	CERN CPU (kHEPSpec 2006)	23	21	21
CERN Disque (PetaBytes)	5.5	6.1	9.1	CERN Disque (PetaBytes)	1.29	1.5	1.7
CERN Bande (PetaBytes)	6.3	6.8	9.6	CERN Bande (PetaBytes)	1.8	2.5	3
T1 CPU (kHEPSpec 2006)	57.6	117	146	T1 CPU (kHEPSpec 2006)	44	65	65
T1 Disque (PetaBytes)	10.8	7.9	13.1	T1 Disque (PetaBytes)	3.29	3.5	3.7
T1 Bande (PetaBytes)	16.3	13	23.6	T1 Bande (PetaBytes)	2.4	3.47	4.42
T2 CPU (kHEPSpec 2006)	89.6	121	140	T2 CPU (kHEPSpec 2006)	38	36	36
T2 Disque (PetaBytes)	12.6	6.6	11.4	T2 Disque (PetaBytes)	0.02	0.02	0.02

Tab. 3.1 (d'après "Official 2011 Experiment Requirements following Scrutiny Group review")

3.1 Planification et allocation de ressources

> Telle est l'inévitable loi : les inégalités sont toujours en proportion inverse de la force de l'autorité publique. Entre le petit et le grand, le pauvre et le riche, c'est cette autorité publique qui rétablit l'équilibre. Si elle fait défaut, il faut de toute nécessité que le faible obéisse au fort, que le pauvre se soumette au riche.
>
> Fustel de Coulanges

Un ensemble fini de ressources indivisibles, identiques et indifférenciables est disponible. Les utilisateurs se partagent les ressources et possèdent chacun les mêmes droits d'usage sur ces ressources. Un gestionnaire a pour objectif le partage équitable de ces ressources entre les différents utilisateurs et selon leurs besoins. On suppose qu'un comité d'évaluation fournit une estimation réaliste des besoins des différents utilisateurs.

Un gestionnaire a la charge de distribuer ces M ressources identiques (appelées ici des machines) à N utilisateurs. Chaque machine allouée à un utilisateur lui est dédiée indéfiniment. Les demandes en machines arrivent au cours du temps de façon croissante à des instants prédéterminés et sont incertaines mais peuvent être anticipées. L'objectif est d'être le plus équitable possible envers ces utilisateurs compte-tenu de leurs demandes.

Un état des lieux des demandes est effectué, éventuellement à intervalles réguliers ou non, à la suite duquel des décisions d'allocation sont prises selon les demandes et la quantité de ressources disponibles à allouer. L'état des lieux recense les demandes des différents utilisateurs et y répond en planifiant les allocations.

Les problèmes de gestion de ressources interviennent lorsqu'une ressource en quantité limitée doit être partagée entre plusieurs participants (des personnes, des groupes ou des entités). Si les ressources doivent toujours être en quantité suffisante quelque soit les besoins des participants on a alors affaire à un problème de dimensionnement. Si par contre les ressources risquent d'être insuffisantes pour satisfaire aux besoins de tous les participants, alors un conflit sur ces ressources doit être résolu : on parle alors de problème d'allocation. Le plus souvent les participants souhaitent que la résolution du conflit sur les ressources se régle de façon équitable, via un médiateur dont la fonction est l'arbitrage de ces ressources.

Exemples de problèmes d'allocation

Voici une liste de différents problèmes d'allocation, en fonction de la ressource partagée :

Allocation de fréquences radio, Orbite de satellites : Le rôle de l'ITU-R est de réguler les plages de fréquences radio et les ressources satellitaires. Pour ces problèmes, l'équité est un objectif [Ahmed et al., 2009]. [Fargier et al., 2004; Bouveret et Lemaître, 2006] résolvent le problème d'allocation équitable et efficace d'un ensemble fini de ressources à l'aide du Leximin, dans le cadre du partage de ressources satellitaires.

Approvisionnement en eau potable : [Wang et al., 2004] où l'approche basée sur le Leximin est utilisée pour allouer équitablement l'eau potable.

L'occupation mémoire : (allocation mémoire entre processus) l'allocation dynamique est l'attribution de mémoire aux différents processus au cours de leur exécution, i.e. la distribution de la

mémoire (ressource limitée) entre les différentes parties de code ou de données. Lorsque la demande excède la capacité, le gestionnaire de la mémoire (allocateur) peut allouer un espace disque (swap) en remplacement mais qui est de moins bonne qualité (vitesse d'accès). Cette mémoire sur disque évolue selon l'utilisation par les processus ; en privilégiant les zones mémoire les plus récemment utilisées par exemple. [Berger et al., 2002] étudie les performances de plusieurs allocateurs de mémoire et propose un allocateur hybride performant. [Wilson et al., 1995] insiste sur le fait que dans les problèmes de l'allocation de mémoire, l'accent a été mis surtout sur les mécanismes d'allocation et pas assez sur les politiques et stratégies d'allocation qui selon les auteurs sont de plus grande importance.

Allocation de bande passante : l'ingénierie des télécommunications a étudié le problème de l'allocation équitable de bande passante dans les flots élastiques [Chiu, 1999; Bertsekas et Gallager, 1987; Ogryczak et al., 2008; Salles et Barria, 2008]. Plusieurs mesures de l'équité sont utilisées avec l'ambition commune de certifier une certaine garantie d'équité [Denda et al., 2000; Allman et al., 1999; Floyd, 2000; Welzl et al., 2010; Kelly et al., 1998; Boudec, 2000].

Le "traffic shaping" est une méthode adoptée par les fournisseurs d'accès aux télécommunications pour garantir une qualité de communication. En effet, une communication massive peut affecter la qualité de l'ensemble des autres communications utilisant le même réseau.

[Meiners et Torng, 2007] analyse le problème de l'ordonnancement de paquets réseaux où les paquets ont différentes qualités de service. Une solution proposée consiste à diviser l'ensemble des paquets en deux groupes : un groupe avec des contraintes de dates butoirs et un autre groupe dont l'objectif est de minimiser la somme des temps de séjour. Le problème est ensuite étudié lorsque les paquets ont tous la même taille.

[Haskins, 2001] déclare que la régulation des flux par un opérateur passe par un cahier des charges ("Service Level Agreements" ou SLA [Naqvi et al., 2008; Macías et al., 2010; Prodan et Wieczorek, 2010]) entre l'opérateur et le client dans lequel les deux parties définissent la qualité de service à fournir, ainsi que par une répartition équitable du débit de sortie. Traditionnellement les limites sont soient des limites dures, soient des limites souples. Une limite dure est une limite stricte que l'utilisateur ne peut pas franchir. Une limite souple est une limite qui ne s'applique que lorsque la capacité est atteinte ; c'est à dire lorsque le lien est saturé. Les limites peuvent concerner les utilisateurs, mais aussi les applications ou se baser sur des priorités entre types de flux. Les limitations peuvent dépendre aussi du moment de la journée.

Ressources physiques : comme nombre de pages d'impression pour une imprimante. [Kay et Lauder, 1988] propose pour la gestion du nombre de pages une méthode dite de quota flottant. Par exemple, une limite absolue est fixée à 10 pages par jour avec récuperation de 2 pages par jour. Si un utilisateur imprime un document de 4 pages, sa limite pour la journée passe à 6 pages. Le jour suivant cette limite passe à 8 pages et augmente ainsi sans jamais dépasser 10 pages par jour.

Allocation d'espace disque : dans un espace de stockage partagé de capacité fixe comme par exemple une baie de stockage. La mise en place de quotas utilisateurs permet d'éviter qu'un groupe d'utilisateur monopolise l'ensemble de l'espace de stockage. Le partitionnement joue le même rôle que les quotas avec une limitation plus rigide. Une fois l'allocation d'une capacité de stockage à un groupe d'utilisateurs effectuée, celui-ci est libre de l'utiliser conformément à la politique d'utilisation de cette ressource. Ainsi du point de vue de l'administrateur des

ressources un bloc a été alloué même si le groupe utilise partiellement cette ressource à un instant donné.

Allocation d'adresses dans un pool d'adresses réseau : l'icann est chargée d'allouer l'espace des adresses de protocole Internet. Une base de données centrale est distribuée sur les registres des différentes régions. A l'échelle d'une organisation, les administrateurs disposent de plages d'adresses IP à distribuer aux différents groupes d'utilisateurs. Il y a actuellement pénurie de l'ensemble des adresses IPV4 disponibles.

Grilles de calcul : un nombre important de groupes (Virtual Organisation) et d'utilisateurs se partagent un ensemble de ressources informatiques mises à disposition par un ensemble de sites. L'objectif est de satisfaire les besoins des groupes et des utilisateurs ; ce qui implique d'avoir un traitement équitable et efficace [Kostreva et al., 2004; Sandmann, 2006; Chevaleyre et al., 2006, 2007]. En effet, des expériences en psychologie [Larson, 1998; Rafaeli et al., 2002; Norman, 2008] ont démontré que lorsque des utilisateurs ou des groupes ressentent un manque d'équité dans le traitement de la part d'un site, ils choisissent de le quitter.

Tout d'abord, [Ernemann et al., 2002] montre l'avantage du partage de tâches dans une grille de calcul dans le sens où les fluctuations de chaque clusters se compensent. Sur plusieurs machines parallèles [Schwiegelshohn et Yahyapour, 2000] propose un algorithme préemptif en ligne qui donne un bon rapport d'approximation à la fois pour le critère du Makespan et de la somme des temps de complétion.

Ce problème a été approché sous le point de vue de la théorie des jeux [Rzadca et al., 2007] où une analogie avec le dilemme du prisonnier est établie. [Tan et Gurd, 2007] ont proposé un mécanisme d'allocation de ressources de calcul, basé sur des processus de marché. L'article suppose des participants non-coopératifs et égocentrés et propose un mécanisme de double enchères pour les départager. Dans [Berten et Gaujal, 2005; Berten et al., 2006; Berten et Gaujal, 2007], un modèle stochastique basé sur les index de Gittins est utilisé afin de répartir au mieux les tâches sur plusieurs clusters.

[Feitelson et Rudolph, 1995; Feitelson et al., 2004] étudient le problème où chaque tâche est en compétition et proposent différentes techniques pratiques mises en œuvre pour résoudre ce problème. [Agnetis et al., 2009] étudie le problème bi-critère où un des agent souhaite minimiser la somme pondérée des dates de fin de ses tâches ; le critère de l'autre utilisateur est borné et devient une contrainte. Il donne une méthode efficace de résolution basée sur la relaxation Lagrangienne. Dans [Mor et Mosheiov, 2010] deux agents sont en compétition pour se partager une machine, les critères utilisés sont basés sur les temps d'avance ("earliness"). Ils proposent un algorithme polynomial pour le critère de minimiser la somme et démontrent que dans le cas où la somme est pondérée le problème devient NP-complet. La complexité des problèmes avec plusieurs agents et une machine est analysée dans [Agnetis et al., 2007].

Lorsque l'ensemble des tâches est divisé en plusieurs ensembles et chaque ensemble a un objectif distinct, on parle aussi de "tâches interférantes". [Balasubramanian et al., 2009] considèrent le problème avec tâches interférantes d'objectifs Makespan et somme des temps de complétion et proposent un algorithme génétique pour ce problème. Le lecteur pourra se rapporter à [Huynh Tuong et al., 2009] pour une bibliographie détaillée sur les problèmes avec tâches interférantes sur une machine.

Dans [Zhao et Sakellariou, 2006], plusieurs DAGs sont ordonnancés sur une grille de calcul avec un souci d'équité, vue ici comme égalité dans les débits de calcul de ces DAGs. [Agnetis

et al., 2010] étudient les ordonnancements multi-agents où chaque agent a son ensemble de tâches et il n'existe pas d'autorité centrale.

Dans [Pascual et al., 2009], les auteurs considèrent un ordonnancement multi-sites et montrent qu'il est toujours possible de produire une solution collaborative qui respecte les objectifs des participants, tout en améliorant globalement les performances du système. Le critère d'évaluation est la date de fin de la dernière tâche (Makespan) pour les tâches appartenant à un site. Ces tâches peuvent s'exécuter sur des sites distants. Un algorithme est proposé qui donne une 3-approximation du makespan optimal. De plus, cet algorithme a la propriété d'inciter les sites à participer. Cependant, les auteurs ne distinguent pas les sites des utilisateurs : en effet, dans le projet EGI par exemple les VOs sont indépendantes des sites fournisseurs de ressources.

L'utilisation répandue de quotas afin de réguler les allocations peut se comprendre comme une conséquence indirecte de la volonté de faire en sorte que chacun puisse utiliser les ressources auquel il a droit sans gêner l'utilisation des autres usagers.

Nous allons maintenant approter plus de détails à l'objectif de nos travaux.

3.1.1 Présentation du problème et définitions

Définition 3.1 (Demande). *Une demande V est un vecteur de $\mathbb{N}^N$ où N est le nombre d'utilisateurs. La i^e coordonnée du vecteur est la demande en nombre de machines du i^e utilisateur.*

Définition 3.2 (Allocation). *Une allocation A est un vecteur de $\mathbb{N}^N$ dont la i^e coordonnée est la part reçue par le i^e utilisateur.*

Chaque utilisateur reçoit au maximum la quantité qu'il a demandé, on doit donc avoir pour une demande V et une allocation correspondante A la propriété suivante : $A \prec_c V$.

Le gestionnaire de ressources doit décider d'allouer plusieurs machines à différents utilisateurs sachant qu'il est limité par la quantité de ressources disponibles : il ne peut pas accorder plus de M machines au total.

Nous pouvons maintenant définir le problème de base de l'allocation de ressources :

Définition 3.3 (Allocation de ressources). *Un problème d'allocation de ressources est défini par N utilisateurs notés $U_1, \ldots, U_N$, une quantité M de ressources homogènes et indivisibles appelées ici machines par commodité. Chaque utilisateur U_k a une demande $V_k \in \mathbb{N}$.*

Une solution pour ce problème est un vecteur d'allocation $A \in \mathbb{N}^n$, tel que $A_k \leq V_k$ est l'allocation perçue par l'utilisateur U_k. La somme des machines allouées doit être inférieure au nombre de machines disponibles : $\sum_{k=1}^{N} A_k \leq M$. L'objectif est de minimiser un critère quelconque, representé sous forme d'un ordre $\prec$ sur ces allocations.

On suppose dans le cas des tâches permanentes qu'une fois que l'utilisateur reçoit A_k machines, il est libre du placement de ses tâches. Cela ne change pas le nombre de machines qu'on lui offre, même s'il ne les utilise pas toutes à un instant donné (à lui d'ordonnancer ses tâches sur les machines proposées comme il l'entend).

Définition 3.4 (Saturation). *Lorsqu'une allocation alloue toutes les machines à sa disposition, on dit que l'allocation est* saturée. *De même, lorsque l'allocation d'un utilisateur est égale à sa demande, on dit que l'utilisateur est saturé.*

3.1.2 Résolution optimale du problème d'allocation équitable dans le cas déterministe

3.1.2.a Water-Filling

Soit le problème de base de l'allocation de ressources avec pour objectif de maximiser l'équité via l'ordre Leximin.

Théorème 3.1 (Allocation équitable de ressources). *Soit V un vecteur de requêtes et soit A^* une allocation maximale pour $<_{Lex}$, alors :*

1. *Si $\sum_{k=1}^{N} V_k \leq M$ alors $A^* = V$*
2. *sinon $\sum_{k=1}^{N} A_k^* = M$ et il existe $\alpha \in \mathbb{N}$ avec $\alpha \geq \lfloor \frac{M}{N} \rfloor$ tel que*
   - *Si $V_k < \alpha$ alors $A_k^* = V_k$*
   - *Si $V_k \geq \alpha$ alors $A_k^* = \alpha$ ou $A_k^* = \alpha + 1$*

   De plus α est calculable en $O(N \log(N))$ tâches.

Démonstration. Les deux premiers points du théorème sont un cas particulier du lemme 2.5, en prenant $X = 0$ et $Y = V$.

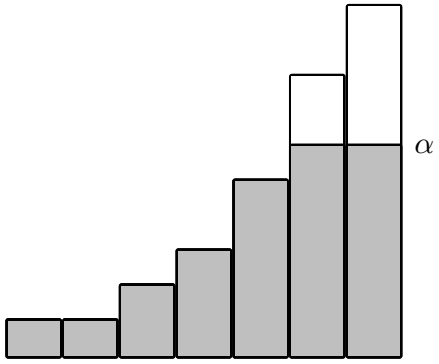


Fig. 3.1.1 Water-filling

Tous les utilisateurs pour lequels $V_i \leq \lfloor \frac{M}{N} \rfloor$ sont saturés. En effet, il est possible d'allouer tous les utilisateurs avec $\lfloor \frac{M}{N} \rfloor$ sans allouer toutes les machines. Ainsi $\alpha \geq \lfloor \frac{M}{N} \rfloor$.

En particulier, nous avons :

$$\sum_{k=1}^{N} \min(\alpha, V_k) \leq M < \sum_{k=1}^{N} \min(\alpha + 1, V_k) \tag{3.1.1}$$

Soit la fonction $M(x) = \sum_{k=1}^{N} \min(x, V_k)$, et soit $u \in \mathbb{R}^+$ tel que $M(u) = M$, alors $\alpha = \lfloor u \rfloor$. $\lfloor u \rfloor$ est calculable en $O(N)$, si le vecteur V est trié. Par conséquent, le temps de calcul est en $O(N \log(N))$. ∎

Remarquons que nous avons redécouvert un algorithme connu sous le nom de *Water-filling* (voir figure 3.1.1). Cet algorithme est utilisé en particulier dans les allocations de flots max-min [Hahne, 1991; Bertsekas et Gallager, 1987]. C'est pourquoi la valeur α porte aussi le nom de "niveau d'eau".

3.1.2.b Cas en ligne avec réallocation de machines : algorithme de "Robin-Hood"

Nous supposerons désormais que les demandes évoluent avec le temps. Dans le cas où le gestionnaire met à disposition les machines qu'il est possible de reprendre en cas de besoin pour les allouer à d'autres utilisateurs, le problème revient à résoudre le cas hors-ligne et à pratiquer des ajustements.

On suppose ici que la réallocation de ressources est autorisée ; lorsqu'une ressource est attribuée on peut la réattribuer quand les demandes évoluent.

Si le nombre de machines demandées est inférieur au total des machines disponibles, il suffit de donner à chacun ce qu'il a demandé, il n'y a pas de conflits donc pas de problème d'équité.

Si le nombre de machines demandés est supérieur au nombre de machines disponibles alors on doit avoir $\sum_{k=1}^{N} A_k = M$ sinon il suffit d'assigner une machine disponible à un utilisateur non saturé.

Ainsi, la politique équitable revient dans ce cas à accepter toutes les demandes jusqu'à saturation, et ensuite à reprendre des ressources pour les redistribuer selon le vecteur obtenu par water-filling.

Montrons qu'il est possible de réattribuer efficacement sans devoir recalculer le niveau d'eau. Il suffit pour cela d'assurer à chaque apparition de tâches la condition du cas hors ligne.

Soit l'algorithme qui donne une machine s'il en reste à chaque demande, et lorsqu'il ne reste plus de machines et si le demandeur est saturé alors il déplace une machine de l'utilisateur qui a le plus de machines vers celui qui en a fait la demande. Dans le cas contraire cela revient à ne rien changer. Appelons cet algorithme "Robin Hood".

Théorème 3.2 (Machines identiques, tâches permanentes, cas en ligne avec réattribution de machines)**.** *L'algorithme "Robin Hood" assure que l'allocation obtenue par chaque utilisateur est à chaque étape une allocation maximale pour le Leximin.*

Démonstration. Soit $V(i)$ le vecteur des demandes et $A(i)$ le vecteur obtenu à la i^e étape. Le vecteur $A(0)$ est le vecteur nul, il est maximum pour le Leximin lorsqu'il n'y a pas de demandes. Supposons que $A(i)$ soit une allocation maximale pour le Leximin pour le vecteur de demandes $V(i)$.

- Si $\sum_{k=1}^{N} V_k(i+1) \leq M$ alors l'algorithme ci-dessus alloue de sorte que $A = V$ et ce vecteur est maximal pour le Leximin.
- Si $\sum_{k=1}^{N} V_k(i) \geq M$ alors on a d'après le théorème 2 :

  $$A_k(i) = \min(V_k(i), \alpha_i) + \epsilon_{i,k}, \epsilon_{i,k} \in \{0,1\}$$

  avec $\sum_{k=1}^{N} A_k(i) = M$.
  Un vecteur maximal pour le Leximin pour le nouveau vecteur de demandes $V(i+1)$ est de la forme :

  $$A_k^* = \min(V_k(i+1), \alpha') + \epsilon'_k, \epsilon'_k \in \{0,1\}$$

  avec $\sum_{k=1}^{N} A_k^* = M$.
  Soit k' l'utilisateur qui a fait la demande.
  - Si k' n'est pas saturé, alors $A_{k'}(i) = \alpha_i + \epsilon_{i,k}$ et en prenant $\alpha' = \alpha$ et $\epsilon' = \epsilon$ on remarque que $A(i)$ est maximal pour le Leximin avec la nouvelle demande soit $A^* = A(i) = A(i+1)$.
  - Si k' est saturé, alors $A_{k'}(i) = V_{k'}(i)$ et $\alpha_i \geq V_{k'}(i)$. Soit k'' l'utilisateur qui a le plus de machines, alors $A_{k''}(i) = \alpha_i + \epsilon_{i,k''}$.

– Si $\epsilon_{i,k''} = 0$ comme k'' est l'utilisateur qui a le plus de machines, c'est que $\forall k, \epsilon_{i,k} = 0$. Prenons pour A^* $\alpha' = \alpha - 1$ et $\forall k \neq k'', \epsilon_{i,k} = 1, \epsilon_{i,k''} = 0$, ce vecteur est obtenu en transférant une machine de k'' vers k et est de la forme A^* donc maximum pour le Leximin.
– Si $\epsilon_{i,k''} = 1$ alors prenons pour A^* $\alpha' = \alpha$ et $\forall k \neq k'', \epsilon_{i,k} = 1, \epsilon_{i,k''} = 0$. De même ce vecteur est obtenu en transférant une machine de k'' vers k et est de la forme A^* donc maximum pour le Leximin.

Par conséquent $A(i+1)$ est maximum pour le Leximin sur le nouveau vecteur de demandes. ■

Etape	U_1		U_2		U_3		U_4		U_5	
	Alloc	*Dem*	*Alloc*	*Dem*	*Alloc*	*Dem*	*Alloc*	*Dem*	*Alloc*	*Dem*
1	4	4	0	0	2	2	0	0	2	2
2	4	4	0	0	3	3	0	0	2	2
3	5	5	0	0	3	3	0	0	2	2
4	**4**	5	**1**	1	3	3	0	0	2	2
5	**3**	5	1	1	3	3	**1**	1	2	2
6	3	6	1	1	3	3	1	1	2	2
7	3	7	1	1	3	3	1	1	2	2
8	3	7	**2**	2	**2**	3	1	1	2	2

Tab. 3.2 Exemple d'application de l'algorithme "Robin Hood" avec 5 utilisateurs et 10 machines

Nous supposons désormais que les allocations s'effectuent sans réallocation, une fois allouée une ressource ne peut pas être allouée à un autre utilisateur.

Lorsque les demandes arrivent et que les allocations sont prises aux points de décisions on parle de *décision en ligne*. Dans ce cas on ne connait pas à l'avance l'ensemble de tous les déroulements possibles et l'évaluation se fait sur le scénario réalisé.

Lorsque les décisions sont prises a priori sur l'arbre de toutes les possibilités d'évolution des demandes on parle de *décision hors ligne*.

3.1.3 Politiques d'allocation avec incertitudes sur les demandes

> The end of law is not to abolish or restrain, but to preserve and enlarge freedom. For in all the states of created beings capable of laws, where there is no law, there is no freedom.
>
> John Locke

Le problème qui nous préoccupe ici est une extension du problème d'allocation de ressources auquel s'ajoute une prise en compte de l'incertitude sur les demandes V et où l'objectif est l'équité sur les allocations.

Dans le cas où les demandes sont soumises à des incertitudes, le responsable doit orienter ses décisions afin d'affronter le risque et l'appréhender. La théorie de la robustesse s'intéresse à réduire

l'impact du risque ou de l'incertitude [Billaut et al., 2005; Artigues et al., 2005, 2008; Esswein et al., 2008].

Les décisions peuvent s'effectuer parmi un ensemble de choix possibles. Un ensemble d'*états* potentiels représente les différentes circonstances pouvant survenir [Roy et Bouyssou, 1993]. Cet ensemble d'états encapsule l'incertitude pouvant se produire. On suppose pour l'instant que l'ensemble de ces états est complet, dans le sens qu'aucun autre état ne pourra subvenir : nous verrons plus loin le cas où l'ensemble des états n'est pas connu dès le départ. Une probabilité peut être éventuellement associée à chaque état de l'ensemble mais cela n'est pas obligatoire car selon le problème il peut être difficile d'évaluer ces probabilités. Quand des probabilités sont associées, on parle de décisions sous risques sinon on parle de décisions sous incertitudes. La donnée d'une décision et d'un état produit un gain quantifiable a priori.

Notons qu'un état peut encapsuler un ensemble d'événements successifs, et que dans ce contexte il faut prendre en compte la succession de ces événements. L'objectif est alors de déterminer la meilleure action (ou suite d'actions) à entreprendre compte-tenu des informations disponibles et selon les différents critères de performance.

Différents modéles d'incertitudes sur les requêtes sont possibles. Une première approche dite réactive est de ne pas faire d'hypothèses sur l'évolution des requêtes ; c'est l'approche choisie pour l'équilibrage de charge en ligne [Azar, 1992]. Cependant, cette approche ne permet pas d'anticiper en tenant compte de certaines informations connues à l'avance. D'autres approches utilisent un modèle probabiliste. Néanmoins, il peut se réveler difficile d'obtenir précisément ces probabilités.

L'alternative que nous avons choisie est de se baser sur un modèle de scénarios : on suppose qu'une planification des futures demandes possibles est disponible au départ et qu'elle contient le scénario qui va se réaliser. Cette approche permet de modéliser des demandes corrélées entre les utilisateurs. Il est aussi possible d'associer une probabilité à la réalisation de chaque scénario, mais ce n'est pas obligatoire.

Les modèles à scénarios sont utilisés couramment dans l'optimisation robuste depuis l'apparition du livre de [Kouvelis et Yu, 1997]. Leur approche se base sur des combinaisons linéaires de moyennes et de variances pondérées par un coefficient d'aversion au risque [Fabozzi et al., 2007]. [Rockafellar et Wets, 1991] définit un opérateur d'agrégation de scénarios via des pondérations linéaires. Dans [Tarhan et Grossmann, 2008] une probabilité est associée à chaque scénario et l'espérance d'un critère est minimalisée. D'autres méthodes d'optimisation stochastique existent [Taflin, 1999].

Le modèle que nous avons retenu combine l'objectif d'équité avec une incertitude sur les requêtes modélisées par des scénarios. A partir d'un planning prévisionnel qui énumère les différents scénarios possibles, l'objectif est de construire une politique hors-ligne qui décide des allocations à attribuer selon le déroulement des scénarios. Pour une telle approche d'optimisation sous incertitudes où une solution est calculée hors-ligne on parle d'optimisation proactive [Billaut et al., 2008].

Prenons un exemple avec 2 utilisateurs : l'un d'eux a demandé toutes les machines et un autre n'a rien demandé pour le moment. Si on donne toutes les machines au premier utilisateur le second n'a plus rien s'il vient à en demander. Par contre, si la politique d'allocation est de ne pas donner toutes les machines, il suffit que le second utilisateur ne demande rien pour avoir une répartition sous-équitable. Par conséquent il n'existe pas de politique idéale ; chaque politique doit établir un compromis. En dépit de cela, nous allons rechercher les meilleures politiques dans un sens à définir.

Comme nous avons affaire à un problème multi-critères on s'attend à ce qu'en général il n'existe

pas une seule politique qui surpasse toutes les autres politiques. Un compromis doit donc être trouvé entre les différents critères. Un des apport de cette thèse est de donner quelques propriétés vérifiées par les politiques Pareto optimales et de donner un algorithme capable de les énumérer.

Nous considérons en effet que chaque scénario correspond à un critère dont la performance est mesurée par l'équité sur ce scénario. Cette méthode a été explorée par [Hites et al., 2006]. Remarquons que la recherche du front de Pareto outrepasse certaines difficultés soulevées dans leur article, car le choix d'une politique parmi toutes les politiques non dominées est laissé à l'appréciation du gestionnaire de ressources ; lequel peut employer des critères supplémentaires propres au contexte pour décider.

3.1.3.a Modélisation par scénarios

Nous supposerons par la suite qu'on ne puisse pas ré-attribuer les ressources : en effet, lorsqu'une ressource est allouée elle ne peut pas en général être restituée pour un autre utilisateur. On cherche des politiques d'allocations qui soient le plus équitable possible, dans un sens à définir. La politique doit attribuer des ressources aux utilisateurs à chaque point de décision. Dans ce contexte, on ne connaît pas parfaitement les futures demandes, mais le gestionnaire peut les anticiper.

Définition 3.5 (Point de décision)**.** *Un point de décision désigne un moment dans la phase en ligne où une décision d'allocation doit être prise dans le cas où si les requêtes ont évoluées. Un point de décision n'est pas obligatoirement associé à un instant précis. Un vecteur de requête est associé à un point de décision.*

Le gestionnaire des ressources est supposé avoir à sa disposition une certaine connaissance des besoins des utilisateurs. Les demandes peuvent être corrélées, par exemple lorsque des équipes travaillent ensemble sur plusieurs projets communs. Lorsque les événements ne sont pas indépendants et que des décisions sont à prendre au fil du temps, on parle de déroulement de scénarios. L'ensemble de ces évolutions est modélisé par un ensemble $\mathcal{S}$ de scénarios.

Définition 3.6 (Scénarios)**.** *Un scenario S est une séquence de points de décision avec leur vecteur de requête correspondant. Cette séquence commence avec le point de décision vide et la demande vide $\big[0 \dots 0\big]$. La dernière demande d'une séquence s'appelle la demande finale.*

On suppose qu'un scénario parmi l'ensemble des scénarios considérés va se réaliser ; l'incertitude repose sur celui qui va se réaliser. Toutefois, l'évolution des demandes au cours du temps permet de connaître un scénario partiel. On définit alors la notion de sous-scénarios :

Définition 3.7 (Sous-scénarios)**.** *S' est un sous-scénario de S si et seulement si la séquence de S' est un préfixe de la séquence de S. On note $S' \subset S$.*

S'il existe un scénario qui contient les points de décision p et p' avec p' avant p, alors on note aussi $p' \subset p$.

Définition 3.8 (Ensemble de scénarios et scénario maximal)**.** *$\mathcal{S}$ est un ensemble de séquences de points de décisions pour le problème d'allocation considéré. Un scénario maximal pour $\mathcal{S}$ est un scénario de $\mathcal{S}$ qui n'est sous-scénario d'aucun autre élément de $\mathcal{S}$.*

Comme supposé précédemment, les requêtes sont croissantes pour chaque utilisateur. Notons que notre modèle peut être aussi utilisé en cas de complète incertitude, dans ce cas l'ensemble des scénarios contient simplement tous les scénarios possibles.

L'ensemble des scénarios peut être représenté comme un arbre. L'arbre associé à $\mathcal{S}$, noté $\mathtt{T}(\mathcal{S})$, est construit de la façon suivante : ses nœuds représentent l'ensemble des sous-scénarios de $\mathcal{S}$ et chaque nœud porte un label, celui ci est le vecteur de requête pour le point de décision de la demande finale du sous-scénario associé. Il y a un arc entre le nœud s_1 et s_2 si et seulement si s_1 est le plus grand sous-sénario de s_2 (i.e. s'il existe un vecteur de requête V tel que $s_2 = \{s_1, V\}$). La demande vide est la racine de l'arbre.

Par la suite l'arbre $\mathtt{T}(\mathcal{S})$ est appelé *l'arbre des possibilités*. Des probabilités peuvent être éventuellement ajoutées sur cet arbre.

L'arbre de scenario peut être vu comme l'arbre des prévisions de toutes les évolutions possibles des demandes. Le problème revient alors à rechercher les politiques les plus équitables pour tous les déroulements de scénarios possibles.

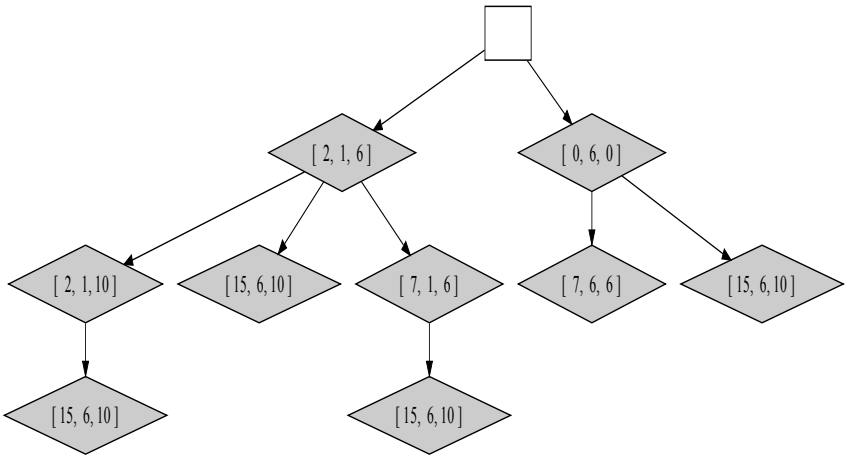


Fig. 3.1.2 Exemple d'arbre des possibilités pour 3 utilisateurs. La requête vide est representée par un carré blanc. Les requêtes sont les labels des nœuds.

Un arbre de possibilités de cardinalité 9 est représenté sur la figure 3.1.2. Le scénario $s_1 = \begin{bmatrix} 2 & 1 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 7 & 1 & 6 \end{bmatrix}$ est un sous-scénario du scénario maximal $s_2 = \begin{bmatrix} 2 & 1 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 7 & 1 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 15 & 6 & 10 \end{bmatrix}$. Remarquons que plusieurs nœuds peuvent porter le même label car la requête est identique mais survient dans des scénarios distincts. Ainsi dans l'exemple, la requête $\begin{bmatrix} 15 & 6 & 10 \end{bmatrix}$ est accessible par plusieurs scénarios.
Remarque importante : dans un arbre des possibilités les dates exactes des points de décisions ne sont pas données. Nous verrons qu'elles ne sont pas nécessaires pour la résolution.

Nous identifions points de décision et nœuds de l'arbre des possibilités.

3.1.3.b Politiques d'allocation

A chaque point de décision une allocation doit être choisie. Celle-ce ne change pas entre les points de décisions. Le problème hors-ligne consiste à attribuer une allocation pour chaque nœud de l'arbre. Ainsi :

Définition 3.9 (Politique d'allocation)**.** *Une* ***politique d'allocation*** $\mathtt{P}$ *associe à chaque point de décision* p *de* $\mathtt{T}$ *une allocation* $\mathtt{P}(p)$ *qui vérifie :*

$$\begin{aligned} \#\mathtt{P}(p) &\leq m \\ \mathtt{P}(p) &\prec_c V^p \\ p' \subset p \Rightarrow \mathtt{P}(p') &\prec_c \mathtt{P}(p) \end{aligned} \tag{3.1.2}$$

Inversement si une fonction vérifie toutes ces propriétés alors elle définit une politique d'allocation. Notons $\mathcal{P}$ *l'ensemble des politiques d'allocation sur* $\mathtt{T}$*.*

L'allocation au point de décision actuel doit être comprise coordonnées par coordonnées entre le vecteur des allocations attribuées au point de décision précédent et le vecteur des demandes actuel.

Considérons encore l'exemple de la figure 3.1.2 avec 10 machines à disposition. Soit la suite de requêtes suivantes : $V^0 \to V^1 \to V^2 \to V^3$ avec $V^0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$, $V^1 = \begin{bmatrix}2 & 1 & 6\end{bmatrix}$, $V^2 = \begin{bmatrix}7 & 1 & 6\end{bmatrix}$, $V^3 = \begin{bmatrix}15 & 6 & 10\end{bmatrix}$. Une politique possible peut être constituée des décisions suivantes : $P(V^1) = \begin{bmatrix}1 & 0 & 5\end{bmatrix}$, $P(V^2) = \begin{bmatrix}3 & 0 & 5\end{bmatrix}$ et $P(V^3) = \begin{bmatrix}3 & 2 & 5\end{bmatrix}$.

3.1.3.c Exemples de politiques d'allocation

Nous donnerons ci-dessous quelques exemples de politiques simples.

Politique de dilapidation Toujours accepter les requêtes des utilisateurs jusqu'à épuisement des ressources :

$$\mathcal{P}[V] = \begin{cases} V, & \text{si } \sum V \leq M, \\ V', & \text{si } \sum V > M, \text{ où } V' \text{ est la première allocation saturée.} \end{cases}$$

Cette politique a le désavantage de ne pas faire d"'économies", dans le sens où les premiers demandeurs peuvent monopoliser les ressources.

Politique parcimonieuse Cette politique n'accepte jamais de donner à un utilisateur plus que la moyenne : $\mathcal{P}[V] = \min_c(V, [\lfloor \frac{M}{N} \rfloor, \dots])$

Cette politique assure une part garantie maximale à chaque point de décision. En effet, elle accorde si et seulement si la demande totale est inférieure à $\lfloor m/n \rfloor$. Toute autre politique soit ne donne pas autant à une demande d'un utilisateur en dessous de $\lfloor m/n \rfloor$, soit au contraire donne trop à un utilisateur et ne peut plus donner jusqu'à $\lfloor m/n \rfloor$ à un autre utilisateur qui en fait la demande.

Pour autant, cette politique est sous-optimale dans bien des cas. Par exemple, si on sait que certains utilisateurs ont une demande qui est toujours inférieure à $\lfloor m/n \rfloor$ toutes les machines ne seront pas utilisées.

Politique proportionnelle $\mathcal{P}[V] = \lfloor \frac{M}{\sum V} V \rfloor$ mais ces décisions d'allocation ne peuvent s'appliquer que si les machines peuvent être réallouées (car les allocations ne sont alors pas forcément croissante sur l'arbre), ce n'est donc pas une politique au sens de la définition.

Etant donné qu'il existe plusieurs mesures possibles pour les politiques et que certaines politiques ne sont pas comparables entre elles, nous allons considérer l'ensemble frontière de Pareto, sur l'ensemble des scénarios possibles.

3.1.4 Algorithme d'énumération des Politiques Pareto pour l'équité

Un des critère pourra être de comparer les allocations dès qu'on atteint la saturation, mais ce critère ne tient pas compte de l'évolution des allocations avant la saturation. En général, on planifie les allocations au moment où les besoins en capacité de stockage ont évolué. C'est pourquoi nous rechercherons les politiques les plus équitables **tout au long des demandes**.

D'autre part, il faut noter qu'une politique décide de son allocation non seulement à partir de la demande actuelle mais aussi des demandes passées (i.e. le chemin pris dans l'arbre des scénarios). En effet, étant donné qu'une allocation ne peut être ré-attribuée à un autre utilisateur, une politique doit tenir compte de cette contrainte pour planifier les allocations en fonction de celles qui sont déjà attribuées.

Notons pour commencer l'observation triviale suivante : Si chaque feuille de l'arbre demande moins de machines au total que le nombre total M alors il suffit de répondre à la demande à chaque étape. Ainsi, il est important de prendre conscience qu'**il n'y a de problème d'équité seulement à partir du moment où il y a potentiellement pénurie des ressources**. Supposons désormais que l'arbre contient des sommets dont la somme des demandes excède la capacité.

3.1.5 Comparaison de politiques d'allocation sous incertitudes

Dans le cas du choix hors-ligne d'une politique d'allocation, on peut associer à chaque point de décision tous les futurs points de décisions possibles avec leurs demandes ainsi que les allocations correspondantes attribuées par une politique d'allocation. Pour comparer deux politiques il faut donc être capable de comparer les deux ensembles d'allocations. Pour cela nous allons comparer les arbres des scénarios possibles dont les nœuds sont valués par les allocations.

Le Leximin est utilisé pour comparer des allocations. Toutefois, nous devons ici comparer des politiques d'allocations. Ainsi, nous allons étendre les ordres Leximin et $\prec_c$ aux politiques :

Définition 3.10 (Comparaison de politiques d'allocation)**.** *Les ordres $\prec_c$ et $\prec_{Lex}$ sont étendus sur les politiques sur $\mathtt{T}$ de la façon suivante :*

- $\mathtt{P} \prec_c^{\mathtt{T}} \mathtt{P}' \Leftrightarrow \forall s \in \mathtt{T}, \mathtt{P}(s) \prec_c \mathtt{P}'(s)$
- $\mathtt{P} \prec_{Lex}^{\mathtt{T}} \mathtt{P}' \Leftrightarrow \forall s \in \mathtt{T}, \mathtt{P}(s) \prec_{Lex} \mathtt{P}'(s)$

On dit que $\mathtt{P}$ est dominée par $\mathtt{P}'$ pour $\prec_c$ (ou $\prec_{Lex}$ respectivement).
Lorsqu'il n'y a pas d'ambiguïtés, l'exposant n'est pas mentionné.

D'après ces définitions, certaines politiques seront incomparables. Afin de prendre une décision, les outils de l'analyse multi-critères seront employés.

Considérons l'exemple de la figure 3.1.3 avec 2 utilisateurs et 8 machines ; l'arbre contient 3 points de décisions avec les demandes associées suivantes : $\{\begin{bmatrix}2 & 5\end{bmatrix}, \begin{bmatrix}5 & 5\end{bmatrix}, \begin{bmatrix}2 & 7\end{bmatrix}\}$. Il y a 6 politiques non dominées pour $\prec_c$, celles-ci sont représentées par un graphe où un arc joint deux politiques si l'une est dominée par l'autre pour $\prec_{Lex}$. Le graphe montre qu'il y a 2 politiques maximales pour $\prec_{Lex}$.

Comme l'exemple le montre $\prec_{Lex}^{\mathtt{T}}$ n'est pas un ordre total dans l'ensemble $\mathcal{P}$ de toutes les politiques, contrairement à $\prec_{Lex}$ sur les allocations.

Une méthode classique employée pour obtenir un ordre total sur $\mathcal{P}$ est d'utiliser une fonction d'agrégation. Cependant une solution maximale pour une fonction d'agrégation quelconque n'est pas forcément maximale pour $\prec_{Lex}^{\mathtt{T}}$. Par exemple : soit la fonction $f_{moyenne}(\mathtt{P}) = \frac{1}{r}\sum_{s\in\mathtt{T}} \mathtt{P}(s)$ et

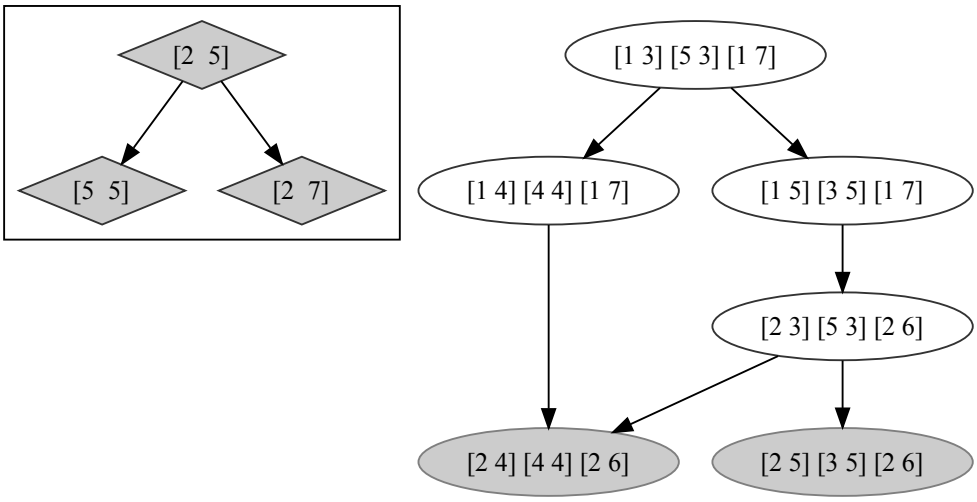


Fig. 3.1.3 Exemple d'arbre des possibilités (cadre à gauche) et graphes correspondant des politiques POC pour 8 machines. Les politiques POE sont en gris clair.

comparons les vecteurs obtenus avec $\prec_{Lex}$. Cette fonction construit des solutions qui ne sont pas maximales, comme le montre l'exemple de la figure 3.1.3. La politique $P = (\begin{bmatrix}2 & 3\end{bmatrix}\begin{bmatrix}5 & 3\end{bmatrix}\begin{bmatrix}2 & 6\end{bmatrix})$ est évaluée par $f_{moyenne}(P) = \begin{bmatrix}3 & 4\end{bmatrix}$ et ce vecteur est maximal sur l'ensemble. Cependant, la politique P est dominée par les deux politiques maximales pour $\prec_{Lex}^{\mathrm{T}}$ sur l'arbre.

Par conséquent, nous nous intéresserons aux politiques maximales pour $\prec_{Lex}^{\mathrm{T}}$ et pour $\prec_{c}^{\mathrm{T}}$:

Définition 3.11 (Politiques Optimales en Coordonnées (POC), Politiques Optimales en Equité (POE))**.** *Une politique est optimale en coordonnées (POC) si et seulement si elle est maximale pour $\prec_{c}^{\mathrm{T}}$. L'ensemble des politiques POC est noté $\mathcal{P}_{\mathrm{T},m}^{c}$ (ou simplement $\mathcal{P}^{c}$ lorsqu'il n'y a pas d'ambiguïtés).*

Une politique est optimale en équité (POE) si et seulement si elle est maximale pour $\prec_{Lex}^{\mathrm{T}}$. L'ensemble des politiques POE est noté $\mathcal{P}_{\mathrm{T},m}^{Lex}$ (ou simplement $\mathcal{P}^{Lex}$ lorsqu'il n'y a pas d'ambiguïtés).

Ainsi, il faut maximiser l'équité tout au long des points de décisions sachant que le vecteur des demandes et celui des allocations sont croissants. Par conséquent, le dilemme consiste dans le fait que d'une part toute décision peut bloquer les décisions suivantes et d'autre part, le fait de ne pas allouer peut devenir sous-optimal.

Remarquons que comme $\prec_{c}$ est un sous-ordre de $\prec_{Lex}$ une politique POE est forcément une politique POC : $\mathcal{P}_{\mathrm{T},m}^{Lex} \subset \mathcal{P}_{\mathrm{T},m}^{c}$.

3.1.6 Caractérisation des politiques POC

A partir du moment où une politique alloue toutes les machines disponibles, elle ne pourra plus allouer d'autres machines pour les demandes suivantes. Ainsi, une politique qui ne donnerait pas toutes les machines sur les feuilles de l'arbre (i.e. à ce moment il est certain que les demandes n'évolueront plus) n'est pas une politique POC. En effet, il suffit de remplacer l'allocation sur une feuille dont la demande totale excède la capacité par une allocation supérieure coordonnée par coordonnée pour améliorer cette politique. Cette remarque est une des propriété qui caractérise les politiques POC comme le montre le théorème suivant :

Théorème 3.3 (Caractérisation des politiques POC)**.** *Soit une requête V sur un nœud de* T*. Une politique* P *est Pareto Optimale en Coordonnées si et seulement si :*

1. *Si V est une feuille de* T *(un nœud sans successeurs) alors :*

$$\begin{cases} \mathtt{P}[V] = V, & Si\ \sum_{i=1}^{N} V_i \leq M \\ \sum_{i=1}^{N} \mathtt{P}[V]_i = M, & Si\ \sum_{i=1}^{N} V_i > M \end{cases}$$

2. *Si V n'est pas une feuille de* T*, soient $V^1, \ldots, V^k$ les successeurs de V alors :*

$$\mathtt{P}[V] = Min_c(V, \mathtt{P}[V^1], \ldots, \mathtt{P}[V^k])$$

où Min_c désigne le minimum pris coordonnée par coordonnée.

Par conséquent, les politiques POC sont caracterisées par les allocations qu'elles donnent sur les points de décisions finales (les feuilles de l'arbre des possibilités).

Démonstration. Soit P une politique POC , alors $\sum_{i=1}^{N} \mathtt{P}[V]_i \leq M$ et $\mathtt{P}[V] \preceq_c V$. D'un autre côté $\mathtt{P}[V]$ doit être plus petit ou égal pour $\prec_c$ que tous ses successeurs directs ainsi : $\mathtt{P}[V] \preceq_c \mathtt{P}[V^j], j = 1 \ldots k$. Ainsi toutes les politiques doivent vérifier :

$$\begin{aligned} \sum_{i=1}^{N} \mathtt{P}[V]_i &\leq M \\ \mathtt{P}[V] &\preceq_c Min_c(V, \mathtt{P}[V^1], \ldots, \mathtt{P}[V^k]) \end{aligned} \tag{3.1.3}$$

Si $\sum_{i=1}^{N} \mathtt{P}[V]_i < M$ et $\mathtt{P}[V] \not\preceq_c Min_c(V, \mathtt{P}[V^1], \ldots, \mathtt{P}[V^k])$ alors il existe un vecteur positif X tel que remplacer $\mathtt{P}[V]$ par $\mathtt{P}[V] + X$ dans P donne une politique valide qui domine P, ce qui est contradictoire. Par conséquent, $\sum_{i=1}^{N} \mathtt{P}[V]_i = M$ ou $\mathtt{P}[V] = Min_c(V, \mathtt{P}[V^1], \ldots, \mathtt{P}[V^k])$.

De plus :

- Lorsque V est une feuille $\sum_{i=1}^{N} \mathtt{P}[V]_i = M$ ou $\mathtt{P}[V] = V$, d'où le résultat. Remarquons que si $\sum_{i=1}^{N} V_i \leq M$ nous devons avoir $\mathtt{P}[V] = V$ et si $\sum_{i=1}^{N} V_i > M$, nous devons avoir $\mathtt{P}[V] = V$.
- Si $\mathtt{P}[V]$ n'est pas une feuille alors si $\sum_{i=1}^{N} \mathtt{P}[V]_i = M$, $\mathtt{P}[V'] = \mathtt{P}[V]$ pour tous les successeurs V' de V.

Par conséquent, une politique POC doit vérifier ces propriétés.

Réciproquement, supposons qu'une politique P verifie ces propriétés. S'il existe une politique P′ qui domine P, alors une telle politique peut-être choisie parmi les politiques POC, ainsi P′ doit vérifier les propriétés ci-dessus. Comme P′ domine P, pour chaque requête V on a $\mathtt{P}[V] \preceq_c \mathtt{P}'[V]$, en particulier sur les feuilles :

- Si $\sum_{i=1}^{N} V_i \leq M$ alors $\mathtt{P}[V] = \mathtt{P}'[V] = V$.
- Si $\sum_{i=1}^{N} V_i > M$ alors $\sum_{i=1}^{N} \mathtt{P}[V]_i = \sum_{i=1}^{N} \mathtt{P}'[V]_i = M$. Ainsi, $\sum_{i=1}^{N} (\mathtt{P}'[V] - \mathtt{P}[V])_i = 0$. Mais comme $\mathtt{P}[V] \preceq_c \mathtt{P}'[V]$, $\mathtt{P}'[V] - \mathtt{P}[V]$ est un vecteur de composantes positives ou nulles. Donc, toutes les différences doivent être égales à zero et $\mathtt{P}'[V] = \mathtt{P}[V]$.

Comme $\mathtt{P}'$ et $\mathtt{P}$ prennent les mêmes valeurs sur les feuilles il s'ensuit, d'après les propriétés, que $\mathtt{P}' = \mathtt{P}$ et donc que $\mathtt{P}$ est une politique POC. ∎

Il existe une méthode souvent utilisée pour l'allocation d'espace disque : c'est la notion de *quotas* ; chaque utilisateur a une limite d'utilisation de son espace disque qu'il ne peut dépasser. On constate donc que cette notion de quotas est une conséquence de l'utilisation équitable [1] des ressources. Notons toutefois qu'une politique qui utilise des quotas arbitraires n'est pas équitable puisqu'il est nécessaire d'imposer aux bornes d'être atteignables afin d'éviter la coercion des ressources.

3.1.7 Propriétés des politiques POE

Commençons par un lemme technique :

Lemme 3.4 (Réallocation de machines croissante pour l'équité)**.** *Soient X, Y des vecteurs tels que $X \prec_c Y$, et soit $Z \in ([X,Y], M)$ (On a $\sum_{i=1}^{N} Z_i = M$).*

Si $Z \prec_{Lex,\neq} Eq([X,Y], M)$, alors il existe des indices i_1 et i_2 tels qu'en réallouant une machine de i_2 à i_1 dans Z on obtienne un vector $Z' \in ((X,Y], m)$ avec $Z \prec_{Lex,\neq} Z'$.

Démonstration. Notons que si $\sum_{i=1}^{N} Z_i = M$ et $Z \neq Eq([X,Y], M)$ alors il existe i_1 tel que $Z_{i_1} < Eq([X,Y], M)_{i_1}$. Grâce au lemme 2.5 nous avons :

- $Y_i \leq \alpha \Leftrightarrow Eq([X,Y], M)_i = Y_i$
- $X_i \geq \alpha + 1 \Leftrightarrow Eq([X,Y], M)_i = X_i$
- sinon $Eq([X,Y], M)_i = \alpha$ ou $\alpha + 1$

Comme $Z_{i_1} < Eq([X,Y], M)_{i_1}$ et $X \prec_c Z$, on a $X_{i_1} < \alpha + 1$ et $Z_{i_1} < \alpha + 1$. Puisque Z et $Eq([X,Y], M)$ utilisent le même nombre de machines, et étant donné que nous avons moins en i_1 dans Z, nous devons avoir dans Z un autre indice i_2 où Z dépasse $Eq([X,Y], M)$. Or nous ne pouvons pas avoir plus que $Eq([X,Y], M)$ si $Y_{i_2} \leq \alpha$ car $Z \prec_c Y$. Ainsi, il faut que $Eq([X,Y], M)_{i_2} \geq \alpha$ et $Z_{i_2} > \alpha$.

Nous avons $Z_{i_1} \leq \alpha < \alpha + 1 \leq Z_{i_2}$.

Si $Z_{i_2} = Z_{i_1} + 1$, alors en réallouant une machine de i_2 vers i_1, nous obtenons un vecteur équivalent pour $\prec_{Lex}$. Mais comme $Z \prec_{Lex,\neq} Eq((X,Y], m)$, il doit exister un autre couple i'_1 et i'_2 tels que $Z_{i'_2} > Z_{i'_1} + 1$. En réallouant de la même manière, nous obtenons un vecteur d'allocation strictement meilleur. ∎

Définition 3.12. *Soit $\mathtt{T}$ un arbre de possibilités de racine V_1 avec les sous-arbres $\mathtt{T}_1, \ldots$. Soit X un vecteur de $\mathbb{N}^N$. Notons $\mathcal{E}(\mathtt{T}, M, X)$ l'ensemble de toutes les politiques sur l'arbre $\mathtt{T}$ qui utilisent au plus M machines et dont les allocations sont supérieures ou égales à X sur V_1.*

Notons $\mathtt{P}^*_{\mathtt{T},m,X}$ la politique POC sur $\mathcal{E}(\mathtt{T}, M, X)$ qui maximisent $\prec_{Lex}$ sur l'ensemble $\mathcal{L}_\mathtt{T}$ des feuilles de l'arbre $\mathtt{T}$: $\forall L \in \mathcal{L}_\mathtt{T}, \mathtt{P}^*_{\mathtt{T},M,X}[L] = Eq([X,L], M)$

1. donc aussi efficace car l'équité est une sous-notion : même sans parler du Leximin nous considérons que l'ordre $\prec_c$ doit être un sous-ordre de tout ordre qui se veut équitable.

Lemme 3.5 (Politique maximale sur les feuilles de l'arbre). $\mathtt{P}^*_{\mathtt{T},M,X}$ *est une politique POE sur* $\mathcal{E}(\mathtt{T}, M, X)$ *et*

$$\mathtt{P}^*_{\mathtt{T},M,X}[V_1] = Min_c(V_1, \underset{L \in \mathcal{L}_\mathtt{T}}{Min_c}(Eq([X, L], M)))$$

Démonstration. Soit Q une politique qui domine $\mathtt{P}^*_{\mathtt{T},m,X}$ sur $\mathcal{E}(\mathtt{T}, m, X)$ pour $\prec_{Lex}$. On peut prendre pour Q une politique POC. Q doit dominer $\mathtt{P}^*_{\mathtt{T},m,X}$ sur toutes les requêtes. Or, comme $\mathtt{P}^*_{\mathtt{T},M,X}$ est maximale sur les feuilles, Q est égale à $\mathtt{P}^*_{\mathtt{T},m,X}$ sur les feuilles. Donc comme Q et $\mathtt{P}^*_{\mathtt{T},m,X}$ sont des politiques POC, $\mathtt{Q} = \mathtt{P}^*_{\mathtt{T},m,X}$. L'expression de $\mathtt{P}^*_{\mathtt{T},m,X}[V_1]$ se déduit alors du théorème de caractérisation des politiques POC. ∎

Théorème 3.6 (Borne inférieure des allocations pour les politiques équitables). *Soit* V_1 *la racine d'un arbre de possibilités* T *et soit* P *une politique POE sur* T *avec* M *machines et avec la contrainte* $\mathtt{P}[V_1] \succ_c X$ *pour un vecteur* X *fixé.*

Alors P *alloue au moins autant de machines sur* V_1 *pour chaque utilisateur que la politique POE qui maximise l'équité sur toutes les feuilles avec la même contrainte :*

$$\mathtt{P}[V_1] \succ_c \mathtt{P}^*_{\mathtt{T},M,X}[V_1]$$

Démonstration. Supposons que $\mathtt{P}[V_1] \not\succ_c \mathtt{P}^*_{\mathtt{T},M,X}[V_1]$, alors il existe un utilisateur i_1 tel que $\mathtt{P}[V_1]_{i_1} < \mathtt{P}^*_{\mathtt{T},M,X}[V_1]_{i_1} \leq V_{1,i_1}$.

Par conséquent, grâce au Théorème 3.3, il existe une requête L sur une feuille telle que :

$$\mathtt{P}[V_1]_{i_1} = \mathtt{P}[L]_{i_1} < \mathtt{P}^*_{\mathtt{T},M,X}[V_1]_{i_1}$$

Or comme :

$$\mathtt{P}^*_{\mathtt{T},M,X}[V_1]_{i_1} = \min(V_{1,i_1}, \min_{L \in \mathcal{L}_\mathtt{T}}(Eq([X, L], M)_{i_1})$$

On en déduit que :

$$\mathtt{P}[V_1]_{i_1} = \mathtt{P}[L]_{i_1} < Eq([X, L], M)_{i_1}$$

Ainsi grâce au lemme 3.4, nous pouvons réallouer une machine dans $\mathtt{P}[L]$ de i_1 vers un autre utilisateur i_2 et la nouvelle allocation est plus équitable. Soit P′ la nouvelle politique obtenue. Cette politique est réalisable car $\mathtt{P}'[V]_{i_2}$ est croissant pour tout prédecesseur V de L. Afin de guarantir ce fait, considérons le premier prédecesseur $\tilde{V}$ de L (peut-être que $\tilde{V} = V_1$) pour lequel le nombre de machines allouées à i_2 est croissant dans P. Il est possible d'échanger une machine de i_2 vers i_1 dans P′ pour chaque requête entre $\tilde{V}$ et L. La politique obtenue est réalisable et domine P, ce qui est contradictoire d'après les hypothèses que P est POE.

∎

Lemme 3.7 (Restriction d'une politique sur un sous-arbre). *Soit* T′ *un sous-arbre de* T *de racine* V' *avec pour successeurs immediats* V *dans* T. *Si* P *est une politique POE sur* T *alors la restriction de* P *sur* T′ *est une politique POE dans l'ensemble des politiques* Q *sur* T′ *vérifiant* $\mathtt{Q}[V'] \succ_c \mathtt{P}[V]$.

Démonstration. Supposons qu'il existe Q qui domine P sur le sous-arbre T′. Si dans P, P est remplacée par Q sur T′, alors la nouvelle politique est réalisable et domine P sur T, ce qui est contradictoire. ∎

Notons que le lemme précédent n'implique pas que la restriction d'une politique POE sur un sous-arbre soit POE sur ce sous-arbre, parmi toutes les politiques sur les sous-arbres (Voir les contre-exemples des figures 3.13 et 3.1.11.a).

Cependant, le lemme peut s'appliquer pour chaque sous-arbre de $\mathtt{T}$, grâce au lemme 3.7 avec $X = \mathtt{P}[V]$. Ce résultat sera utilisé afin d'énumérer l'ensemble des politiques POE d'un arbre donné.

Nous allons maintenant donner le théorème fondamental de caractérisation des politiques POE :

Théorème 3.8 (Caractérisation des politiques POE)**.** *Soit $\mathcal{K}^X_{m(\mathtt{T})}$ l'ensemble de toutes les politiques $\mathtt{P}$ qui utilisent sur chaque requête V un nombre de machines prédéterminé $m(V)$ ($\forall V \in \mathtt{T}, \sum_{i=1}^{N} \mathtt{P}[V]_i = m(V)$ fixé) et qui allouent au moins X ($\forall V \in \mathtt{T}, \mathtt{P}[V] \succ_c X$). Alors, il existe une unique politique $\mathtt{P}$ (à des permutations d'utilisateurs près) qui domine toutes les autres politiques pour $\prec_{Lex}$ dans $\mathcal{K}^X_{m(\mathtt{T})}$. De plus $\mathtt{P}$ vérifie :*

$$\forall V \in \mathtt{T}, \mathtt{P}[V] = Eq([\mathtt{P}[V^-], V], m(V))$$

où V^- est le nœud parent de V (éventuellement vide si V est la racine).

Démonstration. Soit $\mathtt{P}$ une politique de $\mathcal{K}^X_{m(\mathtt{T})}$. S'il existe une requête V pour laquelle $\mathtt{P}[V]$ n'est pas maximale pour $\prec_{Lex}$: $\mathtt{P}[V] \neq Eq([\mathtt{P}[V^-], V], m(V))$, alors il existe des indices i_1 et i_2 d'après le lemme 3.4 tels que la réallocation d'une machine de l'utilisateur i_2 vers l'utilisateur i_1 augmente $\prec_{Lex}$ dans $\mathtt{P}[V]$ avec $\mathtt{P}[V] \prec_c V$.

Nous allons construire une politique $\mathtt{R}$ identique à $\mathtt{P}$ sur toutes les requêtes autres que V et ses successeurs. $\mathtt{R}[V]$ est obtenue en réallouant une machine de l'utilisateur i_2 vers l'utilisateur i_1 dans $\mathtt{P}[V]$.

Pour les successeurs de V, $\mathtt{R}$ prend les mêmes décisions que $\mathtt{P}$ sauf pour l'utilisateur i_1 : si $\mathtt{P}$ alloue une machine à l'utilisateur i_1 $\mathtt{R}$ alloue une machine à l'utilisateur i_2.

Par construction, nous avons $\mathtt{R} \in \mathcal{K}^X_{m(\mathtt{T})}$ et $\mathtt{R}$ est plus équitable que $\mathtt{P}$ sur toutes les requêtes. Par conséquent $\mathtt{P}$ n'est pas POE dans $\mathcal{K}^X_{m(\mathtt{T})}$ et si une politique est POE dans $\mathcal{K}^X_{m(\mathtt{T})}$ alors il faut que

$$\mathtt{P}[V] = Eq([\mathtt{P}[V^-], V], m(V))$$

Ainsi, il existe une unique politique POE dans $\mathcal{K}^X_{m(\mathtt{T})}$ et elle est obtenue itérativement en allouant le vecteur maximal pour $\prec_{Lex}$ pour le nombre de machines fixé et sous la contrainte des allocations passées. ∎

Ainsi, **une politique POE est caractérisée par le nombre de machines allouées à chaque requête**. Une autre propriété importante est que **les politiques POE appliquent l'algorithme du water-filling sur chaque requête sous la contrainte du nombre de machines et de l'allocation précédente**.

Lorsque l'on compare deux politiques avec $\prec_{Lex}$ l'ordre des utilisateurs n'a pas d'importance. Ainsi il serait plus simple de n'utiliser que des politiques où les allocations sont triées. Nous allons montrer par construction que l'on peut se ramener à ce cas. Par conséquent on peut calculer l'ensemble des politiques POE à partir de l'ensemble des politiques POE sur l'arbre des possibilités où les requêtes sont triées.

3.1.7.a Correspondance entre les arbres de possibilités triés et non triés

Rappelons le lemme suivant :

Lemme 3.9.

$$A \prec_c B \Rightarrow \vec{A} \prec_c \vec{B}$$

Ce résultat entraîne récursivement ce lemme :

Lemme 3.10 (Le tri d'un arbre des possibilités est un arbre des possibilités valide.)**.** *Soit un arbre des possibilités* $\mathtt{T}$*, si chaque requête* V^i *est triée alors, le nouvel arbre obtenu est un arbre des possibilités valide i.e. croissant avec* $\prec_c$*. Cet arbre est noté* $\vec{\mathtt{T}}$*.*

Théorème 3.11 (Correspondance entre politiques triées et non triées)**.** *Soit un arbre des possibilités* $\mathtt{T}$ *et soit* $\vec{\mathtt{T}}$ *l'arbre où chaque requête est triée. Alors chaque politique de* $\mathcal{P}^{Lex}_{\vec{\mathtt{T}}}$ *s'obtient à partir d'une politique équivalente pour* $\prec_{Lex}$ *dans* $\mathcal{P}^{Lex}_{\mathtt{T}}$*. De plus, la réciproque est vraie.*

Démonstration. Toutes les politiques considérée dans cette preuve sont des politiques POE.

Soient deux politiques qui sont équivalentes pour $\prec_{Lex}$, cela signifie qu'elles sont identiques à permutation des allocations près.

Si l'arbre des possibilités est composé de requêtes triées, alors il existe toujours une politique dont les allocations sont triées dans l'ensemble des solutions. En effet si on trie toutes les allocations d'une politique, cette nouvelle politique reste valide grâce au lemme ci-dessus et est équivalente à la première politique. Par conséquent, on peut prendre un représentant trié pour chaque politique POE lorsque l'arbre est trié.

Soit une politique $\mathtt{P}$ sur $\mathtt{T}$, si on trie les allocations et les vecteurs de requêtes alors la nouvelle politique est une politique sur $\vec{\mathtt{T}}$ qui est équivalente à $\mathtt{P}$. Ainsi, à partir d'une politique dans $\mathcal{P}^{Lex}_{\mathtt{T}}$ on construit une politique dans $\mathcal{P}^{Lex}_{\vec{\mathtt{T}}}$.

Réciproquement : montrons par induction qu'à partir d'une politique dans $\mathcal{P}^{Lex}_{\vec{\mathtt{T}}}$ on peut construire une politique dans $\mathcal{P}^{Lex}_{\mathtt{T}}$.

Soit $\vec{\mathtt{P}} \in \mathcal{P}^{Lex}_{\vec{\mathtt{T}}}$ et V un nœud de $\mathtt{T}$, de prédecesseur V^- (éventuellement nul). Les nœuds correspondants dans $\vec{\mathtt{T}}$ sont notés $\vec{V}$ et $\vec{V}^-$.

Supposons qu'il existe $\mathtt{P} \in \mathcal{P}^{Lex}_{\mathtt{T}}$ telle que $\mathtt{P}[V^-] \equiv \vec{\mathtt{P}}[\vec{V}^-]$.

Soit $A^- = \mathtt{P}[V^-]$, $\vec{A}^- = \vec{\mathtt{P}}[\vec{V}^-]$, $A = \mathtt{P}[V]$ et $\vec{A} = \vec{\mathtt{P}}[\vec{V}]$.

Nous savons que $\vec{A} = Eq([\vec{A}^-, \vec{V}], m_V)$ avec $m_V = \#\vec{\mathtt{P}}[\vec{V}]$ et $\vec{A}$ est construit par application du water-filling avec $\tilde{m} = m_V - m_{V^-}$ machines supplémentaires. Notons l'étape e $WF(\vec{A}^-, e)$.

Une allocations pour V est construite en appliquant l'algorithme suivant :

Algorithme 1: Construction de $\bar{A}(A, V, \tilde{m})$

$\bar{A} \leftarrow A^-$;
for $e = 1$ **to** $\tilde{m}$ **do**
  Soit k tel que $\bar{A}_k$ soit minimal avec $\bar{A}_k < V_k$;
  $\bar{A}_k \leftarrow \bar{A}_k + 1$;
return A ;

Montrons qu'à la fin de chaque étape e, $\bar{A} \equiv WF(\vec{A}^-, e)$.

Ce résultat est vrai lorsque $e = 0$ par hypothèses sur A^-.

Supposons que cela soit vrai à l'étape $e < \tilde{m}$, soit k tel que $\bar{A}_k$ soit minimal avec $a = \bar{A}_k < V_k$. k existe car $e < \tilde{m}$.

Soit $\mathcal{U} = \{l|\bar{A}_l < a\} \cup \{l|\bar{A}_l = a \text{ et } \bar{A}_l = V_l\}$. Notons que $\bar{A}_l = V_l$ pour chaque élément de $\mathcal{U}$ d'après la définition de k.

Soit $|\mathcal{U}| = c$. Les éléments de $\mathcal{U}$ sont à la position c la plus à gauche dans $WF(\vec{A}^-, e)$. A la position $(c+1)$ dans $WF(\vec{A}^-, e)$ il y a un utilisateur k' avec $WF(\vec{A}^-, e)_{k'} = a$ et $WF(\vec{A}^-, e)_{k'} < \vec{V}_{k'}$. Par conséquent à l'étape $e+1$, l'algorithme du water-filling ajoute une machine à l'utilisateur $\tilde{k} \geq k'$ tel que $WF(\vec{A}^-, e)_{\tilde{k}} = a$.

Cet algorithme ajoute ainsi une machine à l'utilisateur k avec $\bar{A}_k = a$. Ainsi après l'étape $e+1$, $\bar{A} \equiv WF(\vec{A}^-, e+1)$. Récursivement on obtient que $A \equiv \vec{A}$.

Et récursivement sur l'arbre on a $\mathtt{P} \equiv \vec{\mathtt{P}}$. ∎

Remarquons que la preuve ci-dessus est constructive. Ainsi s'il est possible de construire les politiques POE sur l'arbre trié des possibilités alors il est possible d'en déduire toutes les politiques POE sur un arbre équivalent non trié.

3.1.8 Le cas particulier des chaînes

Le cas où l'arbre se réduit à une chaîne de requêtes, i.e. lorsqu'il n'y a pas d'évolutions alternatives, est une situation courante dans la planification des besoins. De plus, rétrospectivement, les requêtes survenues forment une chaîne. Il peut être intéressant de comparer les allocations données avec celles qu'on aurait pu fournir, si l'arbre ne contenait que la chaîne qui s'est réalisée. Dans cette partie, nous nous intéresserons plus particulièrement à ce cas particulier.

Les différents résultats ont une expression plus simple lorsque l'arbre se réduit à une chaîne de requêtes triées par hypothèses, notée $V^1 \to \cdots \to V^r$.

Théorème 3.12 (Politiques POC sur une chaîne)**.** *Pour une chaîne $V^1 \to \cdots \to V^r$, les politiques POC sont de la forme :*

$$\forall i, \mathtt{P}[V^i] = Min_c(V^i, \mathtt{P}[V^r])$$

Dans le cas d'une suite de demandes une politique POC est complètement caractérisée par l'allocation qu'elle alloue pour la dernière demande de la chaîne. C'est à dire que les allocations ne dépendent que de l'allocation sur la requête finale. Cela correspond à la notion de déterminer des quotas *par utilisateur selon leurs besoins.*

Démonstration. Ceci est une conséquence directe du théorème 3.3. ∎

Il nous faut remarquer ici que toutes les politiques basée sur des quotas ne sont pas des politiques POC, en effet il faut pouvoir allouer toutes les machines sur la dernière requête.

Une conséquence importante de ce théorème est que si deux politiques POC (donc en particulier les politiques POE) attribuent la même allocation sur un même point de décision, alors elles ont attribuées les mêmes allocations pour toutes les requêtes précédentes. Cela a pour conséquence que les politiques POC (et POE) peuvent être représentée sous forme d'arbres (voir la figure 3.1.5). Ce résultat sera utilisé pour la construction de ces politiques.

Lemme 3.13 (Extension d'une politique)**.** *Soit $\mathtt{P}$ une sous-politique POE sur la sous-chaîne $V^1 \to \cdots \to V^i, i < r$ avec M machines. Alors $\mathtt{P}$ peut être étendue en une politique POE sur la chaîne complète pour M machines.*

Démonstration. Soit $\mathtt{P}'$ une politique dont $\mathtt{P}$ est une sous-politique. Si $\mathtt{P}'$ n'est pas POE sur la chaîne $V^1 \to \cdots \to V^r$, elle est donc dominée par une politique POE $\mathtt{P}''$. Ainsi la sous-politique $\mathtt{P}''[V^1] \to \cdots \to \mathtt{P}''[V^i]$ domine $\mathtt{P}[V^1] \to \cdots \to \mathtt{P}[V^i]$. Mais comme $\mathtt{P}$ est POE sur la sous-chaîne, il en résulte que $\mathtt{P}[V^1] = \mathtt{P}''[V^1], \ldots, \mathtt{P}[V^i] = \mathtt{P}''[V^i]$, ainsi $\mathtt{P}''$ étend $\mathtt{P}$ en une politique POE sur la chaîne complète. ∎

Nous constatons que la réciproque n'est pas vraie ; une sous-politique d'une politique POE n'est pas en général POE sur la sous-chaîne. Ainsi, le principe de sous-optimalité n'est pas valide ici. Par conséquent, un algorithme basé sur la programmation dynamique, où les politiques POE sont construites itérativement, n'est pas envisageable.

De plus il y a plusieurs façons d'étendre une sous-politique en une politique POE, il suffit que $\mathtt{P}''$ ne soit pas unique dans la preuve ci-dessus.

Définition 3.13 (Politiques Leximin-maximales sur une requête donnée.). *Soit $\mathtt{P}_i^*$ une politique sur $V^1 \to \cdots \to V^i, i \leq r$ telle que :*

$$\mathtt{P}_i^*(V^i) = Eq(V^i, M)$$

$$\forall j \leq i, \mathtt{P}_i^*(V^j) = Min_c(V^j, Eq(V^i, M))$$

Comme $\mathtt{P}_i^*$ est Leximin-maximale sur V^i, $\mathtt{P}_i^*$ est POE sur cette sous-chaîne (conséquence du théorème 3.3). De plus, $\mathtt{P}_i^*$ peut être étendue sur la chaîne complète en une politique POE d'après le lemme précédent.

Lemme 3.14 (Leximin-Décroissance de $\mathtt{P}_i^*(V^1)$). *$(\mathtt{P}_i^*(V^1) = Min_c(V^1, Eq(V^i, M)))_{i=1\ldots r}$ est décroissante pour $\prec_{Lex}$.*

Démonstration. Si la chaîne n'est jamais saturée alors toutes les allocations $(\mathtt{P}_i^*(V^1) = Min_c(V^1, Eq(V^i, m)))$ sont égales à V^1 et le résultat est vrai.

Sinon soit V^i la première requête saturée, alors $(Eq(V^j, m))_{j=i\ldots r}$ est décroissante pour $\prec_{Lex}$ et α_j est décroissant. On a $Eq(V^j, m)_k = V^{j,k}$ si $V^{j,k} \leq h_j$, sinon $Eq(V^j, m)_k = h_j$ ou $h_j + 1$. Ainsi $\mathtt{P}_i^*(V^1) = Min_c(V^1, Eq(V^i, m)) = \min(V^{1,k}, h_j$ ou $h_j + 1)$ car V^i est croissante pour $\prec_c$. Comme α_j décroit, on obtient le résultat. ∎

Pour les chaînes le théorème 3.6 devient :

Lemme 3.15 (Intervalle). *Soit X tel que $X \prec_c V^1$ et $\mathtt{P}$ une politique POE sous la contrainte que $\mathtt{P}[V^1] \succ_c X$, alors*

$$\mathtt{P}[V^1] \succ_c Min_c(Eq([X, V^r], M), V^1)$$

3.1.9 Algorithme de construction des politiques POE sur les chaînes

Théorème 3.16 (Algorithme de construction). *Soit $\mathtt{P}$ une politique POC obtenue en appliquant l'algorithme du water-filling progressivement sur chaque requête de l'arbre. Alors $\mathtt{P}$ est une politique POE.*

Démonstration. Avant de démontrer ce théorème nous avons besoin de ce lemme technique :

Lemme 3.17. *Soient des vecteurs croissants $A \prec_c B \prec_c C$ avec :*

$$A' = Eq([A, C], m_1) \prec_{Lex} B' = Eq([B, C], m_2)$$

alors $A' \prec_c B'$.

Démonstration. Si $A' = B'$ on a le résultat, on suppose maintenant que $A' \neq B'$.

Nous avons $A'_i = \min(C_i, \max(\alpha_i, A_i))$, $B'_i = \min(C_i, \max(\beta_i, B_i))$ avec $\alpha_i = \alpha$ ou $\beta + 1$, $\beta_i = \beta$ ou $\beta + 1$.

Note : Ci dessus c'est vrai aussi si A, B et C ne sont pas triés. Par contre ci-dessous on suppose que A' est trié à cause des indices

Comme $A' \prec_{Lex} B'$, il existe k tel que $\forall i < k, A'_i = B'_i$ et $A'_k < B'_k$. Ainsi, $\forall i < k, \min(C_i, \max(\alpha_i, A_i)) = \min(C_i, \max(\beta_i, B_i))$ et $\min(C_k, \max(\alpha_k, A_k)) < \min(C_k, \max(\beta_k, B_k))$.

Supposons que $\min(C_k, \max(\alpha_k, A_k)) = C_k$, on aurait alors $C_k < \min(C_k, \max(\beta_k, B_k)) \leq C_k$, contradiction. Ainsi, $\max(\alpha_k, A_k) < C_k$. Cela implique donc que $\alpha_k \leq \max(\alpha_k, A_k) < C_k$.

Par conséquent nous avons $\min(C_k, \max(\alpha_k, A_k)) = \max(\alpha_k, A_k) < \min(C_k, \max(\beta_k, B_k))$; comme $\max(\alpha_k, A_k)$ est inférieur au minimum de deux valeurs il est inférieur à chacune d'elles, ainsi $\max(\alpha_k, A_k) < \max(\beta_k, B_k)$, et de même nous avons un maximum de deux valeurs inférieur à une autre valeur donc $\alpha_k < \max(\beta_k, B_k)$.

Or $\forall i, A'_i = \min(C_i, \max(\alpha_i, A_i))$,

- Si $A'_i = A_i$ alors $A'_i \leq B'_i = \min(C_i, \max(\beta_i, B_i))$ car $A_i \leq C_i$ et $A_i \leq B_i$.
- Si $A'_i = \alpha_i$ alors $A'_i \leq B'_i = \min(C_i, \max(\beta_i, B_i))$ car $\alpha_k < C_k$ et $\alpha_k < \max(\beta_k, B_k)$.
- Si $A'_i = C_i$, c'est que $C_i \leq \max(\alpha_i, A_i)$,
  - soit $A_i \geq \alpha_i$ et $C_i \leq \max(\alpha_i, A_i) = A_i$, or $A \prec_c C$, donc $A_i = C_i$, or $A \prec_c B \prec_c C$, donc $A_i = B_i = C_i$.
  - soit $A_i < \alpha_i$ et $C_i \leq \max(\alpha_i, A_i) = \alpha_i$, ou $\alpha_i \geq C_i$ mais comme $\alpha_k < C_k$, c'est que $i < k$ et $A'_i = B'_i$

ainsi $A' \prec_c B'$. ∎

Rappelons que :

$$\mathtt{P}_{i+1} = Eq([\mathtt{P}_i, V^{i+1}], \sum_{k=1}^{N} (\mathtt{P}_{i+1})_k)$$

Soit une autre politique POE Q, selon le théorème 3.8, elle s'obtient aussi de la même façon :

$$\mathtt{Q}_{i+1} = Eq([\mathtt{Q}_i, V^{i+1}], \sum_{k=1}^{N} (\mathtt{Q}_{i+1})_k)$$

Si $\mathtt{P} \prec_{Lex} \mathtt{Q}$, alors si on pose $A = \mathtt{P}_i$, $B = \mathtt{Q}_i$ et $C = V^{i+1}$, le lemme ci-dessus entraine que $\mathtt{P}_{i+1} \prec_c \mathtt{Q}_{i+1}$ et récursivement on obtient que $\mathtt{P} \prec_c \mathtt{Q}$.

Mais comme P et Q sont des politiques POC cela entraine que P = Q.

Récursivement en appliquant le lemme 3.17 on aboutit au résultat. ∎

Remarquons que l'ensemble des politiques POE peut être important pour certaines chaînes :

Théorème 3.18. *Pour la chaîne $\begin{bmatrix} 0 & \dots & 0 & M \end{bmatrix} \to \begin{bmatrix} 0 & \dots & 0 & M & M \end{bmatrix} \to \cdots \to \begin{bmatrix} M & \dots & M \end{bmatrix}$ avec N utilisateurs, $\mathcal{P}^c = \mathcal{P}^{Lex}$ est de cardinalité exponentielle en M.*

Démonstration. Soit P une politique POC sur cette chaîne, à cause du théorème 3.3 appliqué aux chaînes, nous devons avoir $\mathtt{P}_i = \mathtt{P}[V^i] = [0, \ldots, 0, x_{N+1-i}, \ldots, x_N]$ avec $\sum_{i=1}^{N} x_i = M, x_i \geq 0$. Soit Q une autre politique POC. Supposons que P et Q soient triées, cela ne modifie pas la preuve.

Supposons que Q domine P selon $\prec_{Lex}$, alors : $\forall i \in \{1, \ldots, N\}, \mathtt{P}[V^i] \prec_{Lex} \mathtt{Q}[V^i]$ et $\begin{bmatrix} 0 & \ldots & 0 & x_{N+1-i} & \ldots & x_N \end{bmatrix} \prec_{Lex} \begin{bmatrix} 0 & \ldots & 0 & y_{N+1-i} & \ldots & y_N \end{bmatrix}$.

Cela entraîne que pour $i = 1 : x_N \leq y_N$; et pour $i = 2 : x_{N-1} \leq y_{N-1}$; ainsi récursivement $x_i \leq y_i$. Mais comme $\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i = M$ on a $x_i \geq 0, y_i \geq 0$, finalement P = Q.

Pas conséquent, P est une politique POE car elle n'est pas dominée selon $\prec_{Lex}$ pour toutes les autres politiques POC et $\mathcal{P}^c = \mathcal{P}^{Lex}$.

Calculons maintenant la cardinalité de l'ensemble des politiques POC non triées sur cette chaîne. Comme $\{1 + x1, \ldots, i + x1 + .. + xi, \ldots\}$ partitionnent l'ensemble des $M + N$ premiers entiers en $N - 1$ ensembles et réciproquement, un simple calcul donne $|\mathcal{P}^c| = |\mathcal{P}^{Lex}| = \binom{m+n-1}{n-1}$. ∎

Algorithme de construction des politiques POE sur les chaînes

L'algorithme EnumerateAllPOF énumère toutes les politiques POE triées sur une chaîne donnée. Rappelons qu'on peut calculer les politiques non triées à partir de celles triées.

Les principe est le suivant : à chaque étape i, l'algorithme construit des politiques sur la sous-chaîne $V^1 \to \cdots \to V^i, i \leq r$ à partir des politiques construites sur $V^1 \to \cdots \to V^{i-1}$ en utilisant l'algorithme du water-filling. Selon le résultat précédent, cet algorithme construit un arbre dont les nœuds sont des sous-politiques de politiques POE.

Procédure EnumerateAllPOF($V^1, \ldots, V^r, M$)

```
POFSET ← ExtendPolicy(0, 0, V^1) ;
for i = 2 to r do
    NEWSET ← ∅ ;
    forall the P ∈ POFSET do
        NEWSET ← NEWSET ∪ ExtendPolicy(P[V^{i-1}], V^{i-1}, V^i) ;
    end
    POFSET ← NEWSET ;
end
return POFSET ;
```

Procédure ExtendPolicy(A, V, V')

Data : $V^1, \ldots, V^r, M$
SUBPOL $\leftarrow \varnothing$;
LowerBound $\leftarrow Min_c(V', Eq([A, V^r], M))$;
UpperBound $\leftarrow Eq([A, V'], M)$;
$m_{Low} \leftarrow \sum_{i=1}^{N} LowerBound_i$;
$m_{Up} \leftarrow \sum_{i=1}^{N} UpperBound_i$;
$m' \leftarrow m_{Low}$;
repeat
  $A' \leftarrow Eq([A, V'], m')$;
  if $\forall k, A'_k = V_k$ *ou* $A'_k = A_k$ **then**
    $SUBPOL \leftarrow SUBPOL \cup A'$;
    STOP $\leftarrow$ False ;
  end
  else
    STOP $\leftarrow$ True ;
  end
  $m' \leftarrow m' + 1$;
until $STOP$ *ou* $m' > m_{Up}$;
return $SUBPOL$;

Supposons qu'à l'étape $i-1$, toutes les sous-politiques (pour la sous-chaîne $V^1 \to \cdots \to V^{i-1}$) des politiques POE sur la chaîne globale aient été énumérées. Soit P une telle sous-politique.

Grâce au théorème 3.8 nous savons que la prochaine allocation dépend uniquement du nombre de machines attribuées. De plus, nous connaissons une borne inférieure et une borne supérieure sur ce nombre de machines d'après le théorème 3.6.

L'ensemble des allocations pour V^i est construit par la procédure "ExtendPolicy" à partir de $\mathrm{P}[V^{i-1}] : \{Eq([\mathrm{P}[V^{i-1}], V^i], m'), m' = m_{Low}, \ldots, m_{Up}\}$

Finalement, d'après le théorème 3.3 nous savons que pour les politiques POC sur les chaînes, si un utilisateur reçoit strictement moins que sa requête à un point de décision donné alors, il ne recevra plus de machines supplémentaires par la suite. Ainsi, les allocations qui ne vérifient pas cette condition ne sont pas ajoutées à l'ensemble. Grâce au lemme 2.5 $Eq([A, V'], m')$ est $<_c$-croissant selon m' ; par conséquent, la boucle s'arrête dès que la condition est vérifiée.

Ainsi d'après le théorème 3.16, les politiques obtenues sont des politiques POE.

3.1.9.a Complexité de l'algorithme

La procédure "ExtendPolicy" a une complexité de $O(m \times n)$ car le water-filling est de complexité linéaire en n lorsque les vecteurs sont triés et m' est borné par m.

Cette procédure est appelée sur chaque nœud de l'arbre de recherche. Le nombre de nœud est borné par $r \times NbPOF$, où $NbPOF$ est le nombre de politiques POE triées. La complexité de l'algorithme est donc de $O(m \cdot n \cdot r \cdot NbPOF)$.

Le nombre de politiques POE est borné par 2^{m-1} : en effet soit $N(k)$ le nombre maximum de politiques qui peuvent être calculées par nœuds avec k machines non allouées, on a : $N(k) \leq \sum_{j=1}^{k} N(k-j)$, ce qui entraîne le résultat. ($N(k-j)$ est le nombre de politiques pour les nœuds

suivants lorsque j machines ont été allouées)

Dans le cas de la chaîne $\begin{bmatrix}0 & \dots & 0 & M\end{bmatrix} \to \begin{bmatrix}0 & \dots & 0 & M & M\end{bmatrix} \to \dots \to \begin{bmatrix}M & \dots & M\end{bmatrix}$ avec $N \geq M$ utilisateurs, le nombre de politiques triées est égal au nombre de partitions triées de l'entier M. Ce nombre est équivalent à $\frac{e^{\pi\sqrt{2M/3}}}{4M\sqrt{3}}$ [Hardy et Ramanujan, 1918] lorsque $M \to \infty$, et est donc exponentiel en M (rappelons que le résultat précédent donnait le nombre total de politiques non triées).

Cependant le nombre de politiques POE reste en pratique limité, la figure 3.1.4 en donne un exemple.

De plus, dans le cas en ligne, les décisions à prendre à chaque point de décision se limitent à choisir le nombre de machines à allouer et ce nombre se situe entre les deux bornes calculées.

3.1.10 Exemples

Soit la chaîne $V_1 = \begin{bmatrix}0 & 1 & 6\end{bmatrix} \to V_2 = \begin{bmatrix}0 & 6 & 7\end{bmatrix} \to V_3 = \begin{bmatrix}1 & 7 & 8\end{bmatrix} \to V_4 = \begin{bmatrix}4 & 7 & 8\end{bmatrix}$, avec 10 machines (figure 3.1.5).

A la première étape de l'algorithme, l'intervalle pour $\mathtt{P}[V^1]$ est calculé :

- Borne inférieure : $Min_c(V_1, Eq(V_4, m)) = \begin{bmatrix}0 & 1 & 4\end{bmatrix}$
- Borne supérieure : $Eq(V_1, m) = \begin{bmatrix}0 & 1 & 6\end{bmatrix}$

Ainsi, $\mathtt{P}[V^1] \in \{\begin{bmatrix}0 & 1 & 4\end{bmatrix}, \begin{bmatrix}0 & 1 & 5\end{bmatrix}, \begin{bmatrix}0 & 1 & 6\end{bmatrix}\}$

A la seconde étape, 8 sous-politiques sont calculées, 2 d'entre elles sont saturées (toutes les machines ont été allouées). Au cours de la troisième étape, à partir de $\begin{bmatrix}0 & 2 & 6\end{bmatrix}$ nous avons l'intervalle $[\begin{bmatrix}1 & 2 & 6\end{bmatrix}, \begin{bmatrix}1 & 3 & 6\end{bmatrix}]$ mais l'utilisateur 2 ne peut pas être augmenté (théorème 3.3). Par conséquent la sous-politique $\begin{bmatrix}1 & 3 & 6\end{bmatrix}$ n'est pas ajoutée. La même situation apparaît lorsqu'on considère $\begin{bmatrix}0 & 4 & 4\end{bmatrix}$ ($\begin{bmatrix}1 & 4 & 5\end{bmatrix}$ n'est pas ajoutée). Notons qu'il existe dans cet exemple 2 politiques qui ne maximisent pas l'équité sur un des nœud.

3.1.11 Enumération des politiques POE pour les arbres

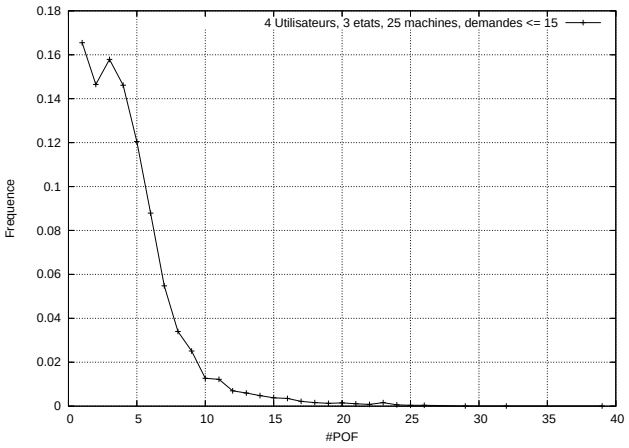
Les théorèmes 3.6 et 3.8 sont toujours applicables pour les politiques sur les arbres. Par conséquent l'algorithme présenté pour les chaînes peut s'utiliser sur les arbres, cependant le théorème 3.16 ne se généralise pas : les politiques calculées ne sont pas nécessairement toutes POE. Il faut donc rajouter une phase de filtrage qui ne conserve que les politiques non dominées dans l'ensemble.

3.1.11.a Quelques contre-exemples pour les arbres

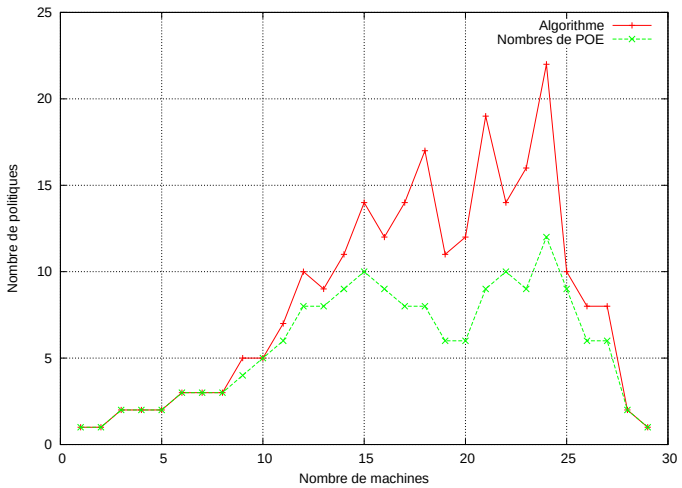
La restriction d'une politique POE sur une branche n'est pas toujours une politique POE. En effet, soit `T` un arbre des possibilités et `C` une branche de `T`. Alors la restriction d'une politique POE sur `T` à `C` n'est pas forcément une politique POE sur `C` comme le montre la figure 3.1.6.

De plus, 2 politiques POE peuvent amener à une allocation identique à partir de différents chemins, comme le montre l'exemple de la figure 3.1.7. Ainsi, les politiques POE sur les arbres ne partagent pas les mêmes allocations précédentes lorsqu'elles ont la même allocation pour une requête donnée.

Sur la figure 3.1.7 il n'existe que 5 politiques POE cependant, l'algorithme énumère 15 politiques. Comme le montre la figure 3.1.4(b), en pratique le nombre de politiques énumérées reste



(a) Nombre empirique de politiques POE et leur fréquence respective pour 6 utilisateurs, 30 machines, et les chaînes composées de 3 requêtes.



(b) Nombres de politiques énumérées et nombre de politiques POE pour un arbre fixe de 14 nœuds avec 3 utilisateurs et un nombre variable de machines.

Fig. 3.1.4 Résultats expérimentaux sur le nombre de politiques POE.

toutefois raisonnable.

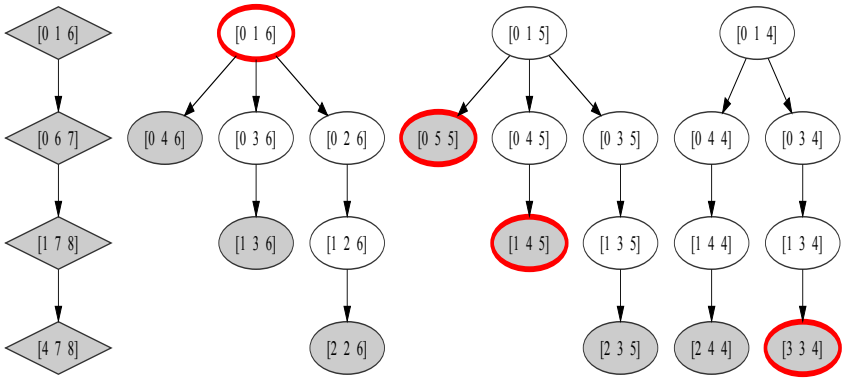


Fig. 3.1.5 Exemple de la construction de l'ensemble des politiques POE. La chaîne est situé à gauche (losanges). Les allocations saturées sont en gris. Les politiques P^* sont marquées avec une bordure plus épaisse. Les 8 politiques POE sont énumérées par l'algorithme

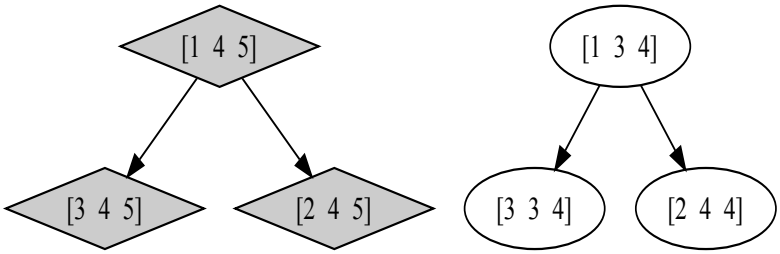


Fig. 3.1.6 $m = 10$, $C = [1\,4\,5] \to [2\,4\,5]$. La politique $[1\,3\,4] \to [2\,4\,4]$ n'est pas POE sur C car elle est dominée par $[1\,4\,4] \to [2\,4\,4]$.

3.1.12 Politiques d'allocation en ligne sans restitution de machines

Dans le cas en ligne où les demandes sont totalement inconnues l'arbre de décision est complet : il contient toutes les demandes possibles. Nous allons maintenant étudier le problème en ligne sous l'aspect suivant : on cherche les politiques qui seront non dominées sur le scénario effectivement pris et inconnu au départ. L'évaluation des politiques s'effectue alors *a posteriori*.

Le problème de décision en ligne est différent de celui hors ligne ; en effet, une politique en ligne suppose qu'il existe une seule chaîne ,inconnue du gestionnaire au début, mais qui va se réaliser. De plus, les politiques seront comparées entre elles avec cette chaîne (inconnue au départ) alors que dans le cas hors ligne, on effectue la comparaison sur l'arbre de toutes les demandes possibles.

D'après le Théorème 3.8, les allocations à chaque point de décision dépendent seulement du

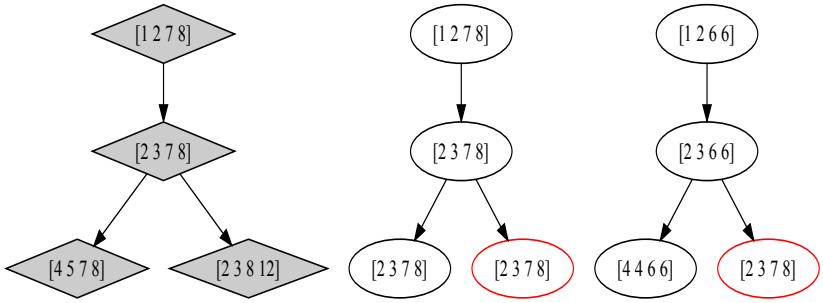


Fig. 3.1.7 $m = 20$, Ces deux politiques POE partagent l'allocation $[2\,3\,7\,8]$.

nombre de machines qu'on décide d'allouer et de l'allocation précédente.

Dans le cas de politiques purement en ligne on compare les politiques avec la chaîne de points de décisions réalisée ; laquelle est inconnue au départ. La différence avec le cas hors-ligne est qu'on suppose qu'une chaîne de requêtes va se réaliser et que **l'évaluation des politiques se fera seulement sur cette chaîne**, et non pas sur toutes les alternatives de l'arbre des possibilités comme dans le cas hors-ligne. Dans le cas en ligne, nous n'avons aucune information sur la chaîne qui va se réaliser. Autre différence avec le cas hors-ligne : les décisions se font au fur et à mesure que les points de décisions surviennent. Dans le cas hors-ligne, une politique tenant compte de tous les scénarios possibles est choisie au départ et n'est plus modifiée par la suite.

Etant donné que la politique recherchée doit être POC, cette politique doit se baser sur des quotas. Cependant en ligne rien n'oblige à ce qu'a priori les quotas soient décidés dès le départ : à un instant donné on connaît une borne inférieure du quota final ; celle-ci peut se préciser avec le temps et selon les demandes des utilisateurs.

L'algorithme d'énumération sur les chaînes peut encore être utilisé dans ce cadre, à condition de connaître la dernière requête $V_{\max}$. C'est à dire *une borne atteignable sur les demandes de tous les utilisateurs*. En effet, à chaque point de décision l'algorithme calcule l'intervalle du nombre de machines à allouer en prenant en compte $V_{\max}$. Dans le cas en ligne, le scénario est inconnu au départ mais se réalise au fur et à mesure. Il suffit donc d'exécuter cet algorithme :

Algorithme 2: Allocation en ligne $A(A^-, V, m')$

input : A^- désigne l'allocation précédente
input : V est le nouveau vecteur de demandes
input : m' est le nombre de machines à allouer

LowerBound $\leftarrow Min_c(V', Eq([A, V^r], M))$;
UpperBound $\leftarrow Eq([A, V'], M)$;
$m_{Low} \leftarrow \sum_{i=1}^{N} LowerBound_i$;
$m_{Up} \leftarrow \sum_{i=1}^{N} UpperBound_i$;
/* Le gestionnaire doit fixer la valeur de m' telle que: */
/* Il faut que m' se situe dans l'intervalle $[m_{Low}, m_{Up}]$. */
$m_{Low} \leq m' \leq m_{Up}$;
/* Il faut que m' vérifie la condition des politiques POC sur A: */
$A = Eq([A^-, V], m')$;
$\forall h, A_h^- < V_h \Rightarrow A_h = A_h^-$;

return A ;

La politique construite sera au final une politique POE sur la chaîne réalisée. De plus, contrairement à l'énumération complète de toutes les politiques POE sur l'arbre de possibilités, ce calcul est rapide à réaliser.

Enfin comme la politique obtenue est POE, elle est aussi POC. Dans le cas des chaînes cela signifie qu'elle correspond à une partition des ressources. Notons qu'avec l'approche en ligne cette partition n'est pas détérminée a priori mais se précise en fonction des besoins et des décisions prises, ce qui fournit plus de souplesse au gestionnaire et aux utilisateurs.

3.1.13 Remarques

Nous avons présenté un nouveau problème d'allocation équitable où les ressources sont allouées indéfiniment. Nous avons défini les politiques d'allocations sur l'arbre des possibilités et les critères de comparaison entre ces politiques. Ceci nous a conduit aux notions de politiques optimales en coordonnées et optimales en équité. Nous avons ensuite étudié les propriétés de ces politiques en vue de les caractériser. De ces propriétés nous avons énoncé un algorithme d'énumération dans le cas où l'arbre des possibilités se réduit à une chaîne et un algorithme en ligne correspondant en a été déduit. L'algorithme reste applicable pour le cas des arbres, mais nous n'avons plus la garantie que l'ensemble énuméré contient seulement les politiques POE. De plus, dans le cas des arbres nous avons une explosion combinatoire du nombre de politiques.

Cet algorithme en ligne peut s'utiliser dans tous les problèmes où il faut décider d'une répartition des ressources entre utilisateurs de mêmes droits et où l'équité est un critère important.

Pour le cas hors-ligne, le gestionnaire pourra par exemple choisir ,parmi toutes les politiques POE, celles qui dominent le plus de politiques POC dans l'ensemble. Dans l'exemple de la figure 3.1.3 la politique POE $((2,4)(4,4)(2,6))$ est celle qui répond à ce critère. Cependant le choix d'une politique à appliquer dépend grandement de la situation : toutes les politiques POE ne sont pas pertinentes selon le problème. Par exemple dans certains cas de figure, il serait inacceptable de laisser certains utilisateurs sans machines, même si une demande de leur part est très peu probable. Dans d'autres cas il ne serait pas admissible de laisser des ressources inutilisées en cas de demandes

ultérieures. Comme ces objectifs sont en conflit des critères additionnels doivent être mis en place pour préciser les intentions des participants et ainsi guider le choix du gestionnaire.

Il est important de garder en mémoire que l'évaluation des besoins doit aussi être impartiale et fiable. En effet, il est facile pour un utilisateur de sur-estimer ses besoins afin d'obtenir une plus grande part des ressources. Des mécanismes de paiement peuvent être mis en place dans un deuxième temps afin d'éviter ces abus [Nielsen, 1970; Papadimitriou et Valiant, 2009].

3.2 Tâches permanentes avec machines hétérogènes

Supposons que les tâches sont permanentes (de durées infinies) et que n utilisateurs demandent chacun n_i services et reçoient $m_i \leq n_i$ machines. Le gestionnaire a à sa disposition m machines de vitesses différentes. La métrique par utilisateur est le débit de calcul : la somme des vitesses des machines obtenues. Dans le cas de machines identiques ceci est équivalent au nombre de machines obtenues comme précédemment. Il s'agit de distribuer les machines de façon équitable entre les utilisateurs.

Avec des machines de vitesses différentes, il s'agit de répartir les machines sachant qu'on ne peut pas donner à un utilisateur plus de machines qu'il a de services (tâches permanentes).

Notons que :

Lemme 3.19. *Seules sont utilisées les $\sum n_i$ machines les plus rapides.*

Démonstration. En effet si ce n'est pas le cas une machine non utilisée est plus rapide qu'une des machine utilisée. Il suffit alors de les remplacer pour améliorer la part d'un utilisateur. ∎

Lemme 3.20. *S'il existe des utilisateurs qui ne demandent qu'un seul service ($n_i = 1$) alors la machine de vitesse maximale est allouée à l'un d'eux.*

Démonstration. Supposons que la machine de vitesse maximale $v^{\max}$ soit allouée à un utilisateur qui a plusieurs machines, et soit un utilisateur avec $n_i = 1$ et une machine de vitesse v. Alors, si on échange la machine de cet utilisateur avec la machine de vitesse maximale l'équité augmente car $\min(v^{\max} + \ldots, v) = v$ et $\min(v + \ldots, v^{\max}) > v$. ∎

Théorème 3.21 (Sous-répartition équitable)**.** *Soit $V = (V_1 \ldots V_n)$ une répartition maximale pour l'équité, pas forcément triée. Alors la répartition $(V_1 \ldots V_{k-1} V_{k+1} \ldots V_n)$ est maximale pour l'équité pour le sous-problème où sont supprimés de l'ensemble de départ les machines allouées à l'utilisateur k.*

Démonstration. Si $(V_1 \ldots V_{k-1} V_{k+1} \ldots V_n)$ n'était pas maximale pour l'équité on aurait une répartiton $(V'_1 \ldots V'_{k-1} V'_{k+1} \ldots V'_n)$ avec $(V_1 \ldots V_{k-1} V_{k+1} \ldots V_n) \prec_{Lex} (V'_1 \ldots V'_{k-1} V'_{k+1} \ldots V'_n)$ et si on lui adjoint V_k le vecteur $(V_1 \ldots V_n)$ ne serait plus maximal pour l'équité. ∎

On en déduit que les utilisateurs qui ne demandent qu'une tâche permanente ont les machines les plus rapides de l'ensemble. Pour ce qui est des autres utilisateurs on peut plonger le problème du $P||C_{\max}$ dans ce problème.

3.2.1 Machines hétérogènes avec tâches permanentes.

Si les machines sont hétérogènes, la mesure par utilisateur devient la somme des vitesses des machines obtenues. Le problème de l'équité revient à $P||C_{\min}$ [Azar et Epstein, 1997; Woeginger, 1997] dès qu'il y a au moins deux utilisateurs. En effet, on recherche à partitionner l'ensemble des machines aux utilisateurs de sorte que le minimum de la somme des vitesses d'un utilisateur soit maximal.

Supposons que chaque machine autorise le partage de processus (*processus sharing*). Notons v_j la vitesse de la machine j et $\beta_{i,j} \in 0,1$ la part donnée par la machine j à l'utilisateur i alors on a à résoudre un système linéaire. Toutes les machines sont utilisées pleinement et chaque utilisateur a au plus n_i machines.

Soit $\gamma_{i,j}$ une variable binaire qui vaut 1 si l'utilisateur U_i a une tâche sur la machine j et 0 sinon :

$$\max_{\prec_{Lex}}[\sum_{j=1}^{m}\beta_{1,j}v_j,\ldots,\sum_{j=1}^{m}\beta_{i,j}v_j,\ldots]$$

$$\begin{cases} \beta_{i,j} & \in & [0,1] \\ \gamma_{i,j} & \in & \{0,1\} \\ \forall j \in [1,M], \sum_{i=1}^{n}\beta_{i,j} & = & 1.0 \\ \beta_{i,j} & \leq & \gamma_{i,j} \\ \forall i \in [1,n], \sum_{j=1}^{m}\gamma_{i,j} & \leq & n_i \end{cases}$$

Dans ce chapitre, nous avons étudié des problèmes d'allocation de ressources. Dans la suite de cette étude, nous aborderons des problèmes d'ordonnancement équitable de tâches à durée limitée.

Chapitre 4

Équité et ordonnancement

La recherche en ordonnancement est un domaine de la Recherche Opérationnelle. Elle consiste à étudier les façons d'organiser et de planifier l'exécution d'opérations au cours du temps de façon à respecter un ensemble de contraintes, éventuellement en cherchant à maximiser un objectif.

4.1 Complexité des problèmes d'ordonnancement multi-utilisateurs

Afin de classifier les différents problèmes d'ordonnancement, nous allons présenter la notation proposée par Graham, Lawler, Lenstra et Kan [Graham et al., 1979] et étendre celle-ci afin de traiter le cas des problèmes multi-utilisateurs.

Définition 4.1 (Notation de Graham). $\alpha|\beta|\gamma$

- α *désigne l'environnement d'exécution.*
  Par exemple :
  - **1** *: Problème à une machine.*
  - **P** *: Problème à* m *machines parallèles identiques.*
  - **Q** *: Problème à* m *machines parallèles de vitesse différente.*
- β *désigne les propriétés ou contraintes des tâches ainsi que les contraintes sur les ressources de traitement.*
  Par exemple :
  - pmtn *: les tâches peuvent être interrompues.*
  - prec *: les tâches ont des contraintes de précédence entre elles.*
  - r_i *: chaque tâche possède une date de disponibilité.*
  - $p_j = 1$ *ou* $p_j = p$*, lorque les durées des tâches sont constantes.*

  Nous ajoutons à ces notations classiques la propriété suivante :
  - ***unique*** *: chaque utilisateur ne possède qu'une seule et unique tâche.*
- γ *désigne habituellement la fonction d'objectif à optimiser.*
  Par exemple :
  - $C_{\max}$ *("Makespan") : date de fin d'exécution de l'ensemble des tâches.*
  - $\sum C_i$ *: somme des temps de complétion des tâches.*
  - $\sum F_i = \sum(C_i - r_i)$ *: somme des temps de séjour avec* $F_i = t_i + p_i - r_i$*).*
  - $F_{\max} = \max_i(C_i - r_i)$ *: temps de séjour maximal.*
  - D_i *: le débit d'une tâche est le rapport entre sa quantité de calcul et son temps de séjour (*$\frac{p_i}{F_i}$*).*
  - S_i *("Stretch") : le ralentissement d'une tâche (ou "stretch" ou "slowdown" en anglais [Bender et al., 1998; Bansal et al., 2004]) est l'inverse du débit.*
  - *Si chaque tâche a une date échue* $\bar{d}_i$*, notons* $L_i = t_i + p_i - \bar{d}_i$ *le retard d'une tâche.*

Nous avons alors les critères suivants : $L_{\max}$ le retard maximal et $\sum L_i$ la somme des retards. De plus notons U_j qui vaut 1 si la tâche ne respecte pas sa date échue et dans le cas contraire 0. Le critère $\sum U_j$ représente le nombre de tâches en retard.

Notons N le nombre d'utilisateurs. Chaque utilisateur h a K_h tâches de durées $p_1^h, p_2^h, \ldots, p_{K_h}^h$. La date de début d'exécution de la tâche p_j^h est notée t_j^h. Notons $C_{\max}^h = \max_{1 \leq j \leq K_h} t_j^h + p_j^h$ la date de fin de la dernière tâche de l'utilisateur h.

Nous proposons d'étendre cette notation aux problèmes multi-utilisateurs de la façon suivante : notons $\gamma_{\prec}$ où γ désigne la fonction de comparaison entre les utilisateurs. Les valeurs de γ forment un vecteur dont les indices sont les utilisateurs. Ce vecteur est appelé le *vecteur d'évaluation.* $\prec$ est l'ordre utilisé pour comparer ces vecteurs d'évaluation.. Par exemple $[C_{\max}^h]_{\prec_c}$ dénote le problème de la recherche des ordonnancements non dominés pour le vecteur de critère du Makespan par utilisateur. Lorsqu'il n'y a pas d'ambiguïtés on note $[C_{\max}^h]_{\prec_{Lex}}$ au lieu de $[-C_{\max}^h]_{\prec_{Lex}}$ pour le problème de minimisation itéré du Makespan des utilisateurs.

Notation 4.1 (Quantité de calcul)**.** *Notons $P^h = \sum_{j=1}^{K_h} p_j^h$ la quantité de calcul de l'utilisateur h, c'est la somme des durées des tâches de l'utilisateur h.*

4.1.1 Prélude à l'étude de la complexité

Voici quelques résultats connus de complexité :

$P||\sum C_j$ est polynomial et résolu par les ordonnancements SPT [Horn, 1973; Bruno et al., 1974]

$1|r_i|\sum C_i$ est NP-difficile d'après [Lenstra et al., 1977]. Ainsi, le problème $1|unique, r_i|\sum C^h$ où chaque utilisateur n'a qu'une tâche l'est aussi.

$P||C_{\max}$ est NP-complet au sens fort [Lenstra et al., 1977].

$P|r_j|F_{\max}$ est une généralisation de $P||C_{\max}$, il est donc aussi NP-complet au sens fort.

$1|r_j|\max S_j$ est NP-complet [Bender et al., 1998].

$1|r_j|\sum S_j$ est NP-complet [Legrand et al., 2008].

On peut réduire le problème du Makespan sur M machines dans un problème périodique. Soit une instance du problème du "Minimum Makespan Scheduling" constituée de M machines identiques et de N tâches de durées $p_1, \ldots, p_N$. Considérons le problème suivant avec M machines, un seul utilisateur ayant N tâches périodiques de période $\alpha = \sum_{i=1}^{N} p_i$ et de durées $p_1, \ldots, p_N$. Alors minimiser le temps de présence de cet utilisateur revient à minimiser le Makespan du premier problème. Comme "Minimum Makespan Scheduling" est NP-Difficile, le problème de minimiser les temps de présence pour des tâches périodiques (ou non d'ailleurs) l'est aussi.

Ainsi, en général un problème non périodique difficile l'est aussi dans le cas périodique (il suffit d'ajuster avec une période assez grande et de recopier les tâches du problème non périodique à chaque période. La recherche d'une solution périodique pour le problème périodique donne alors une solution du problème non périodique). Par contre, l'inverse n'est pas forcément vrai, si un problème est polynomial dans le cas non périodique, on ne peut a priori rien en conclure pour le cas périodique.

4.2 Problèmes à une machine

Etudions quelques problèmes multi-utilisateurs sur une machine.

4.2.1 Critère du Makespan

Dans cette partie, nous étudions le problème multi-utilisateurs où le vecteur d'évaluation est le vecteur des dates de fin de la dernière tâche pour chaque utilisateur. Notons ce vecteur $[C^h_{\max}]$.

Le principe d'évaluation basé sur le Makespan établit qu'entre deux utilisateurs celui qui est le moins bien desservi est celui dont la date de fin de la dernière tâche est la plus grande. Ce principe d'évaluation a du sens, lorsque les dates de disponibilité des tâches sont considérées comme des retards et que seul compte la date où toutes les tâches d'un utilisateur sont terminées.

4.2.1.a Le problème $1||[C^h_{\max}]_{\prec_{Lex}}$

Les utilisateurs se partagent une seule ressource non divisible et les tâches sont toutes disponibles au départ. L'objectif est de placer les tâches de chaque utilisateur sur la machine de sorte que le vecteur d'évaluation $[-C^h_{\max}] = [-C^1_{\max}, \ldots, -C^N_{\max}]$ soit maximal pour l'ordre Leximin. Nous allons démontrer que ce problème est de complexité polynomiale.

Définition 4.2 ("Shortest Total Processing Time" (STPT))**.** *Une politique est STPT si elle exécute les tâches des utilisateurs dans l'ordre de leur quantité de calcul. Les tâches de l'utilisateur qui a la quantité de calcul la plus faible sont exécutées d'abord, et ainsi de suite pour les tâches des utilisateurs suivants.*

Théorème 4.1. *Les politiques STPT (pour "Shortest Total Processing Time") sont optimales pour le problème* $1||[C^h_{\max}]_{\prec_{Lex}}$.

Démonstration. A la fin d'une tâche on peut exécuter immédiatement après la tâche suivante car toutes les tâches sont disponibles à la date 0. Ne pas exécuter cette tâche immédiatement ajoute un retard à toutes les tâches suivantes et déteriore donc le vecteur d'évaluation $[-C^1_{\max}, \ldots, -C^N_{\max}]$. Ainsi, il n'y a pas de temps d'oisiveté entre deux tâches sur la machine : par conséquent les ordonnancements semi-actifs sont donc dominants.

Considérons un utilisateur h et supposons qu'il existe un ensemble de tâches appartenant à d'autres utilisateurs entre deux tâches $O1$ et $O2$ de l'utilisateur h. Étant donné que toutes les tâches sont disponibles à la date 0 il est possible de permuter l'ensemble de tâches avec la tâche $O1$. Cela ne modifie pas le Makespan de l'utilisateur h et avance toutes les tâches de l'ensemble, donc améliore (ou laisse inchangé) le Makespan de tous les utilisateurs de cet ensemble. Ainsi regrouper les tâches par utilisateur est dominant.

Si on échange deux utilisateurs successifs $h1$ et $h2$ les dates des débuts d'exécution des autres utilisateurs ne changent pas. Soit t la date de début d'exécution de la première tâche avant $h1$ et $h2$, exécuter $h1$ suivi de $h2$ donne $C_{h1} = t + P^{h1}$ et $C_{h2} = t + P^{h1} + P^{h2}$, exécuter $h2$ suivi de $h1$ donne $C_{h2} = t + P^{h2}$ et $C_{h1} = t + P^{h1} + P^{h2}$. Ainsi, il est plus équitable selon $\prec_{Lex}$ d'exécuter $h1$ d'abord si $P^{h1} \leq P^{h2}$.

Par échanges successifs on obtient un ordonnancement qui consiste à exécuter les groupes de tâches de chaque utilisateur dans l'ordre des P^h croissants. Ces politiques ont toutes le même vecteur d'évaluation. Nous notons ces ordonnancements STPT pour "Shortest Total Processing Time". Inversement si une politique n'est pas STPT, elle peut être améliorée par l'échange précédent.

■

La figure 4.2.1 montre un ordonnancement SPT avec 3 utilisateurs, le vecteur d'évaluation correspondant est $[32, 24, 17]$. Un ordonnancement STPT est montré sur la figure 4.2.2 avec pour

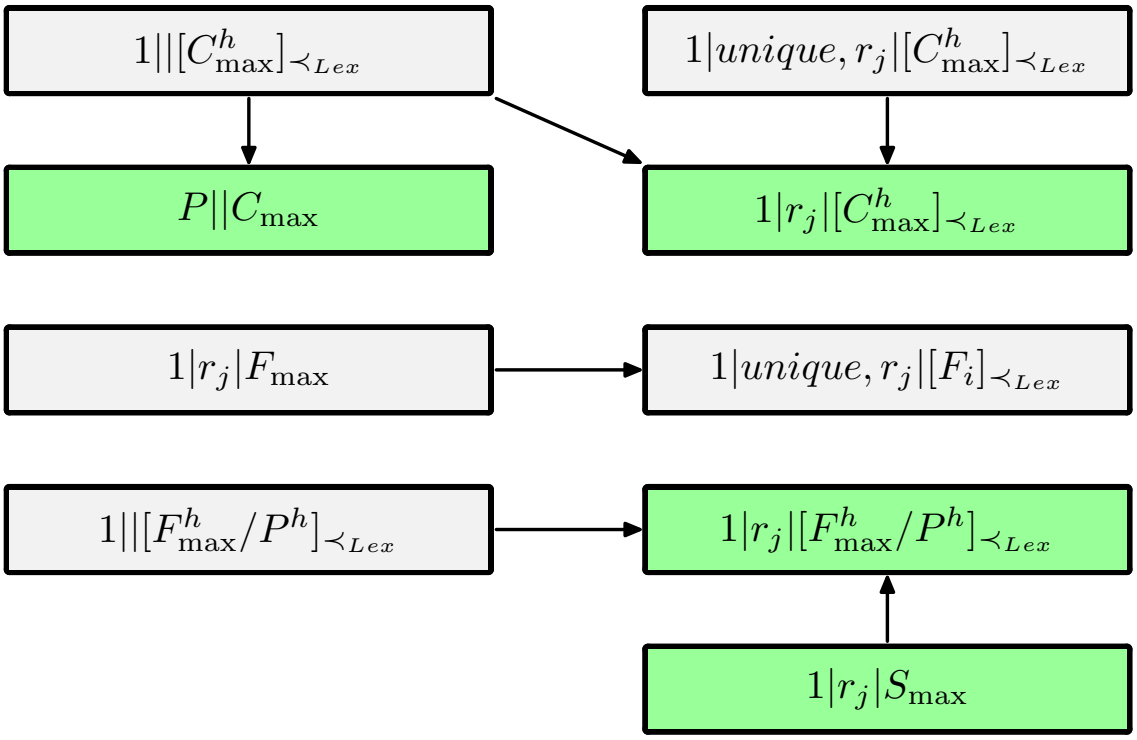


Fig. 4.1.1 Réductions de différents problèmes d'ordonnancements évoqués ci-dessous. Problèmes NP-Complet : en vert, Problèmes polynomiaux : en gris.

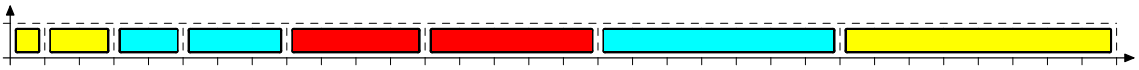


Fig. 4.2.1 Exemple d'ordonnancement SPT. Vecteur d'évaluation : $[32, 24, 17]$

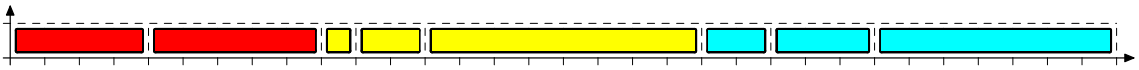


Fig. 4.2.2 Exemple d'ordonnancement STPT. Vecteur d'évaluation : $[20, 32, 9]$

vecteur d'évaluation $[20, 32, 9]$. On constate que d'une part prendre en compte uniquement la notion de tâche et non d'utilisateur peut pénaliser les utilisateurs et d'autre part, que les politiques STPT généralisent les politiques SPT à la notion d'utilisateur.

Remarquons que dès qu'il y a plusieurs machines le problème devient NP-complet car on retrouve $P||C_{\max}$ même dans le cas où il n'y a qu'un utilisateur. Voici une heuristique utilisable pour $P||[C^h_{\max}]_{\prec_{Lex}}$. Lorsqu'on considére le dernier utilisateur il est possible de caler toutes ses tâches à la suite pour que la dernière de ses tâches sur chaque machine se termine à la date de $C^h_{\max}$. Considérons maintenant l'ensemble des tâches du dernier utilisateur. Alors le problème de maximiser le $C_{\min}$ de ces tâches donne une borne pour le Makespan du second utilisateur. L'heuristique procède alors ainsi : Pour chaque utilisateur appliquer LPT sur m machines pour obtenir une valeur heuristique de $C^h_{\max}$ par utilisateur. On trie alors les utilisateurs pas $C^h_{\max}$ croissants puis on applique LPT successivement sur cette liste d'utilisateurs.

4.2.1.b Le problème $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$

Nous allons montrer que ce problème est NP-difficile au sens fort. Nous nous basons sur la réduction au problème de décision suivant :

Définition 4.3 (3-PARTITION). *Soit un ensemble de $3n$ entiers $\{a_1, \ldots, a_n\}$, de somme Bn. Existe-t-il une partition de cet ensemble en n ensembles disjoints, chacun contenant 3 éléments, de sorte que la somme de chaque ensemble est égale à B ?*

Ce problème est NP-difficile au sens fort [Garey et Johnson, 1975] et il l'est encore même si on impose que $\frac{B}{4} < a_i < \frac{B}{2}$. Définissons le problème de décision pour $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$:

Définition 4.4. *Soit N utilisateurs, V un vecteur de $\mathbb{R}^N$ et N' tâches. Chaque tâche i appartient à un utilisateur $U(i)$, elle a une durée p_i, elle est disponible à la date r_i et s'exécute à la date t_i sur la machine. $C^h_{\max}$ désigne le Makespan de l'utilisateur h avec $C^h_{\max} = \max_{i|U(i)=h} t_i + p_i$. Existe-t-il un ordonnancement tel que $[C^h_{\max}] \prec_{Lex} V$?*

Théorème 4.2. *Le problème $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$ est NP-difficile au sens fort.*

Démonstration. Nous allons réduire le problème 3-PARTITION dans $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$.

Soit un ensemble de $3n$ entiers $\{a_1, \ldots, a_n\}$ de somme Bn, avec $\frac{B}{4} < a_i < \frac{B}{2}$. Construisons le problème suivant : soit n utilisateurs, le premier utilisateur possède $3n$ tâches de durées $\{a_1, \ldots, a_n\}$ et disponibles à la date 0. Les utilisateurs $2 \leq h \leq n-1$ possèdent chacun une seule tâche de durée 1 et elle est disponible à la date $(h-1)B$. Prenons $V = [(B+1)n - 1, B+1, \ldots, (h-1)B+1, \ldots]$.

S'il existe un ordonnancement tel que $[C^h_{\max}] \prec_{Lex} V$ alors cet ordonnancement cale toutes les tâches des utilisateurs $2 \leq h \leq n-1$ à leur date de disponibilité (voir la figure 4.2.3). De plus, on a $C^1_{\max} \leq (B+1)n - 1$, Étant donné que la somme des durées des tâches est de $(B+1)n - 1$, on a $C^1_{\max} = (B+1)n - 1$. Ainsi, entre deux de ces utilisateurs les tâches du premier ont une durée totale de B et comme $\frac{B}{4} < a_i < \frac{B}{2}$ il ne peut y avoir que 3 de ses tâches.

Par conséquent le problème de décision 3-PARTITION avec $\{a_1, \ldots, a_n\}$ a une solution si le second problème en a une. Ainsi, $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$ est de complexité NP difficile au sens fort.

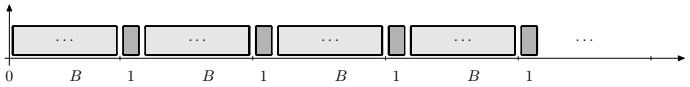


Fig. 4.2.3 Réduction de 3-PARTITION à $1|r_i|[C^h_{\max}]_{\prec_{Lex}}$.

■

4.2.1.c Le problème $1|unique, r_i|[C^h_{\max}]_{\prec_{Lex}}$

Lorsque chaque utilisateur n'a qu'une tâche, nous allons montrer que le problème devient polynomial.

Chaque tâche est calée sur la fin d'une autre tâche ou sur sa date de disponibilité. En effet si une tâche n'est pas calée sur la date de fin d'une autre tâche ni sur sa date de disponibilité alors on peut avancer cette tâche et le vecteur des Makespan pour ce nouvel ordonnancement domine l'ancien.

Définition 4.5 (Blocs de tâches)**.** *Un bloc de tâches est une suite de tâches telles que*
- *la première tâche est calée à sa date de disponibilité*
- *les autres tâches sont calées sur la date de fin de la tâche précédente.*

Par conséquent chaque tâche appartient à un bloc de tâches. Les tâches forment des blocs dont la première tâche d'un bloc est une tâche disponible le plus tôt dans le bloc. Cela signifie que dès que des tâches sont disponibles l'une d'elles est sélectionnée pour s'exécuter.

Dans le cas où chaque utilisateur n'a qu'une tâche le problème se simplifie. Considérons l'ordonnancement obtenu en exécutant les tâches dans l'ordre de leur arrivée (FIFO), cet ordonnancement est de Makespan optimal pour la dernière tâche, noté $C^{\texttt{FIFO}}_{\max}$. Étant donné que chaque utilisateur ne possède qu'une tâche l'utilisateur de la dernière tâche n'a pas d'autres tâches dans l'ordonnancement. Ainsi minimiser le Makespan de l'avant-dernier utilisateur revient à minimiser la date de fin de l'avant-dernière tâche.

Les tâches du dernier bloc doivent rester en bloc pour que le dernier Makespan soit minimal. En effet comme la première tâche du dernier bloc est calée sur sa date de disponibilité et que la date de fin de la dernière tâche doit rester inférieure au Makespan $C^{\texttt{FIFO}}_{\max}$ obtenu par l'ordonnancement FIFO, il ne peut y avoir d'oisiveté sans dégrader la date de fin de la dernière tâche.

Définition 4.6 (Inversion du temps)**.** *Soit un ordonancement O, et O' l'ordonnancement obtenu en inversant le sens du temps d'origine τ : les durées des tâches restent inchangées, les deadlines des tâches deviennent des dates de disponibilités et inversement.*

$$\begin{aligned} p'_i &= p_i \\ t'_i &= \tau - (t_i + p_i) \\ d'_i &= r_i \\ r'_i &= d_i \end{aligned}$$

Considérons l'ordonnancement obtenu en inversant le sens du temps avec $\tau = C^{\texttt{FIFO}}_{\max}$, de sorte que les dates de disponibilités des tâches du bloc deviennent des deadlines ($d_i = C^{\texttt{FIFO}}_{\max} - r_i$).

Selon [Jackson, 1955] si un ordonnancement respectant les deadlines existe alors l'ordonnancement obtenu en exécutant les tâches dans l'ordre de leur deadline (EDF pour "Earliest Deadline First") est un ordonnancement qui les respecte. Notons ici que l'ordre EDF dans l'ordonnancement où le temps est inversé correspond à l'ordre FIFO. Ainsi si les dernières tâches sont placées l'ordre FIFO nous permet de savoir s'il existe ou non un ordonnancement de toutes les autres tâches. Comme on cherche à minimiser successivement les Makespans des tâches, il suffit de trier les tâches du dernier bloc par durée décroissante, ensuite pour chaque tâche on la place en dernière position puis on exécute les tâches restantes dans l'ordre FIFO : si l'ordonnancement obtenu

respecte les choix effectués c'est qu'il existe au moins un ordonnancement avec cette tâche en dernière position. Comme on souhaite minimiser le Makespan de l'avant-dernière tâche, il faut placer la tâche de plus grande durée possible à la fin. On sélectionne donc cette tâche grâce à la méthode décrite ci-dessus. S'il existe plusieurs tâches de même durées on sélectionne la tâche de disponibilité la plus grande.

Ainsi, on applique cette sélection pour les tâches suivantes du bloc et ainsi de suite de blocs en blocs. L'ordonnancement obtenu est alors par construction optimal en équité pour le Makespan dans le cas d'une tâche par utilisateur et l'algorithme est polynomial.

On remarquera que dans le cas où toutes les tâches du bloc sont disponibles à la même date, on retrouve l'ordre STPT.

Exemple

Prenons l'exemple suivant avec un utilisateur par tâche :

Utilisateur	Disponibilité	Durée
1	0	3
2	6	3
3	8	2
4	10	1
5	14	1
6	14	3
7	14	3
8	18	1
9	20	1
10	22	2

L'ordonnancement $FIFO$ donne un ordonnancement optimal pour le Makespan du dernier utilisateur : $(1)(2,3,4)(5,6,7,8,9,10)$ qui est de $14+1+3+3+1+1+2=25$

Trions les tâches du dernier bloc par durées décroissante et en cas d'égalité par disponibilité décroissante : $6,7,10,5,8,9$.

Commençons par placer la tâche en dernière position 6, les autres en FIFO avant cette tâche. S'il n'y a pas d'oisiveté engendré sur la machine l'ordonnancement est valide. $(5,7,8,9,10,6)$ n'est pas valide car il engendre une oisiveté entre la tâche 8 et la tâche 9. Prenons la tâche suivante 7 : $(5,6,8,9,10,7)$ de même n'est pas valide pour la même raison, pour la tâche 10 : $(5,6,7,8,9,10)$ l'ordonnancement est valide et on place donc la tâche 10 à la fin.

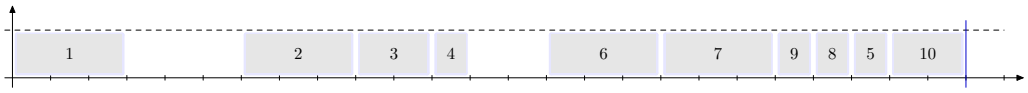


Fig. 4.2.4 Exemple d'ordonnancement optimal pour $1|unique, r_i|[C^h_{\max}]_{<Lex}$.

En continuant avec les autres tâches du bloc, on obtient finalement l'ordonnancement suivant pour le dernier bloc : $(6,7,9,8,5,10)$ et on poursuit de même pour les autres blocs pour obtenir l'ordonnancement équitable en Makespan : $(1)(2,3,4)(6,7,9,8,5,10)$ représenté sur la figure 4.2.4

4.2.2 Critère du temps de séjour

Le principe d'évaluation dans cette partie est la comparaison des temps de séjour ; ce principe s'applique lorsque les dates de disponibilité ne correspondent pas à des retards et lorsque la durée des tâches ne rentrent pas en jeu dans la comparaison.

4.2.2.a Le problème $1|unique, r_i|[F^i]_{\prec_{Lex}}$

Nous allons montrer que ce problème est de complexité polynomiale. Commençons par un résultat classique sur le problème de la minimisation des temps de séjour :

Théorème 4.3 (FIFO est optimale pour $1|r_i|F_{\max}$)**.**

Le problème de maximiser l'équité en temps de séjour sur une machine avec des dates de disponibilité est résolu par FIFO-SPT (exécuter les tâches dans l'ordre de leurs arrivées et en cas d'égalité exécuter la tâche de plus petite durée) :

Théorème 4.4 (FIFO-SPT)**.** *FIFO-SPT est optimale pour le problème* $1|unique, r_i|[F^i]_{\prec_{Lex}}$.

Démonstration. Soit un ordonnancement avec deux tâches exécutées à la suite, notées 1 et 2. Si $r_1 = r_2$ alors le temps de séjour maximal ne change pas, il faut exécuter d'abord la tâche de durée minimale. Les autres tâches demeurent inchangées.

Supposons que $r_1 < r_2$. Notons (a) le cas où la tâche 1 est exécutée avant la tâche 2 et (b) le cas où la tâche 2 est exécutée avant la tâche 1.

Dans le cas (b) on a $F^2_{(b)} = C^2_{(b)} - r_2 < C^2_{(b)} - r_1 < C^1_{(b)} - r_1 = F^1_{(b)}$. Ainsi $F^1_{(b)}$ est le temps de séjour maximal entre les tâches 1 et 2 ($F^1_{(b)} = \max(F^1_{(b)}, F^2_{(b)})$).

Or dans le cas (a) la date de fin de la tâche 1 qui est $C^1_{(a)}$ ne peut que diminuer ($C^1_{(a)} < C^1_{(b)}$) et donc son temps de séjour $F^1_{(a)} < F^1_{(b)}$ aussi. De plus la tâche 2 ne peut que se terminer avant la date de fin de la tâche 1 dans le premier ordonnancement : $F^2_{(a)} \leq C^1_{(b)} - r_2 \leq F^1_{(b)}$. Ainsi le maximum des temps de séjour a diminué. Par ailleurs, les tâches finissent plus tôt, ce qui ne peut que diminuer les temps de séjour des tâches suivantes.

Par conséquent, l'équité augmente lorsqu'on échange les tâches 1 et 2 et si un ordonnancement n'est pas FIFO il est dominé par un autre ordonnancement. Un ordonnancement optimal pour ce problème est donc FIFO-SPT. ∎

4.2.3 Critère du temps d'attente

4.2.3.a $1|unique, r_i|[t_i - r_i]_{\prec_{Lex}}$

La complexité de ce problème demeure inconnue, nous proposons ici quelques règles de dominance.

Remarquons que pour ce problème si $r_1 = r_2$ alors l'attente de la première tâche à s'exécuter ne change pas et il faut donc en premier lieu exécuter la tâche de plus courte durée.

Supposons que $r_1 < r_2$

$2 \to 1$

$$t_2 = \max(r_2, t)$$
$$C_2 = t_2 + p_2$$
$$W_2 = t_2 - r_2 = \max(0, t - r_2)$$

$$t_1 = \max(r_1, C_2) = C_2 = t_2 + p_2$$
$$W_1 = t_1 - r_1 = t_2 + p_2 - r_1 = \max(p_2 + r_2 - r_1, p_2 + t - r_1)$$
$$r_1 \leq r_2 \Rightarrow (t - r_2 \leq p_2 + t - r_1) \text{ et } 0 \leq p_2 + r_2 - r_1 \Rightarrow W_2 \leq W_1$$
$$C = t_1 + p_1 = \max(r_2 + p_1 + p_2, t + p_1 + p_2)$$

$1 \to 2$

$$t'_1 = \max(r_1, t)$$
$$C'_1 = t'_1 + p_1$$
$$W'_1 = t'_1 - r_1 = \max(0, t - r_1)$$

$$t'_2 = \max(r_2, C'_1) = \max(r_2, r_1 + p_1, t + p_1)$$
$$W'_2 = t'_2 - r_2 = \max(0, r_1 - r_2 + p_1, t - r_2 + p_1)$$
$$C' = t'_2 + p_2 = \max(r_2 + p_2, r_1 + p_1 + p_2, t + p_1 + p_2)$$
$$C' \leq C$$

$$t - r_1 < p_2 + t - r_1 \Rightarrow W'_1 < W_1$$

$$\begin{cases} W_1 &= p_2 - r_1 + \max(r_2, t) \\ W_2 &= \max(r_2, t) - r_2 \\ W'_1 &= \max(r_1, t) - r_1 \\ W'_2 &= \max(0, \max(r_1, t) + p_1 - r_2) \end{cases}$$

Comme $C' \leq C$, en exécutant 1 avant 2 les tâches suivantes sont avancées. Une condition suffisante pour améliorer l'équité globalement est que $\max(W_1, W_2) > \max(W'_1, W'_2)$ ou $\max(W_1, W_2) = \max(W'_1, W'_2)$ et $\min(W_1, W_2) > \min(W'_1, W'_2)$.

- $t \leq r_1$

$$\begin{cases} W_1 &= p_2 - r_1 + r_2 \\ W_2 &= 0 \\ W'_1 &= 0 \\ W'_2 &= \max(0, p_1 + r_1 - r_2) \end{cases}$$

Comme $r_1 < r_2$, $W_1 > 0$ on cherche les conditions pour que $\max(W_1, W_2) = W_1 > \max(W'_1, W'_2) = W'_2$, soit :

$$\begin{array}{lcl} p_2 - r_1 + r_2 & > & p_1 + r_1 - r_2 \\ p_1 + 2r_1 & < & p_2 + 2r_2 \end{array}$$

Et réciproquement.

- $r_1 \leq t \leq r_2$

$$\begin{cases} W_1 &= p_2 - r_1 + r_2 \\ W_2 &= 0 \\ W_1' &= t - r_1 \\ W_2' &= \max(0, t + p_1 - r_2) \end{cases}$$

Comme $r_1 < r_2$, $W_1 > 0$ et $\max(W_1, W_2) = W_1$, de même comme $t \leq r_2$ et $p_2 > 0$, $W_1' < W_1$. Ainsi, il reste à comparer W_1 avec W_2'. On ne peut pas améliorer l'équité si $W_1 = W_2'$ car $W_1' > W_2 = 0$. Par conséquent, il faut avoir $W_1 > W_2'$, soit comme $W_1 > 0$:

$$\begin{array}{lcl} p_2 - r_1 + r_2 & > & t + p_1 - r_2 \\ p_1 + r_1 + t & < & p_2 + 2r_2 \end{array}$$

Et réciproquement.

- $t \geq r_2$

$$\begin{cases} W_1 &= p_2 - r_1 + t \\ W_2 &= t - r_2 \\ W_1' &= t - r_1 \\ W_2' &= t + p_1 - r_2 \end{cases}$$

Puisque $p_1 > 0$, $W_2' > W_2$, de même $W_1' < W_1$ et comme $r_1 < r_2$, $W_2 < W_1'$. On en déduit que $\max(W_1, W_2) = W_1$. Il reste à comparer W_1 avec W_2' :

$$\begin{array}{lcl} p_2 - r_1 + t & > & t + p_1 - r_2 \\ p_1 + r_1 & < & p_2 + r_2 \end{array}$$

On déduit ces règles de dominance pour ce problème :

Théorème 4.5 (Dominance). $1|unique, r_i|[t_i - r_i]_{\prec Lex}$

- *Si la tâche i qui minimise $r_i + p_i$ est telle que $r_i \leq t$ il faut l'exécuter. Si $r_i > t$, alors la tâche i domine les tâches j avec $r_j > r_i$.*
- *Si aucune tâche n'est disponible avant t et si la première tâche disponible minimise $2r_i + p_i$ alors il faut l'exécuter.*

Considérons l'exemple suivant :

	1	2	3	4	5
Disponibilité r_i	0	4	8	10	11
Durée p_i	7	2	8	1	6
$p_i + r_i$	7	6	16	11	17
$p_i + 2r_i$	7	10	24	21	28

Nous déduisons du théorème précedant que

- Avant la date $t = 0$ aucune tâche n'est disponible, la tâche 1 minimise $p_i + 2r_i$, il faut alors l'exécuter.
- A la date $t = 7$, la tâche qui minimise $p_i + r_i$ est la tâche 2, il faut l'exécuter.

– A la date $t = 9$, la tâche qui minimise $p_i + r_i$ est la tâche 4 mais $r_4 > 9$. On sait que la tâche 4 domine toutes les tâches suivantes, ici la tâche 5. Il faut donc choisir entre la tâche 3 et la tâche 4. Ici l'ordonnancement qui est le plus équitable pour le critère des temps d'attente sur les 4 premières tâches est $(1, 2, 4, 3)$ mais pour les 5 tâches l'ordonnancement le plus équitable est $(1, 2, 3, 4, 5)$.

4.2.4 Le problème $1|unique, r_i, \bar{d_i}|[L^i]_{\prec_{Lex}}$

$1|r_i|L_{\max}$ est un problème fortement NP difficile [Sadykov et Lazarev, 2005], par conséquent $1|unique, r_i, \bar{d_i}|[L^i]_{\prec_{Lex}}$ l'est aussi. [Lazarev et al., 2007] a developpé un schéma d'approximation pour le problème $1|r_i|L_{\max}$.

Le cas multi-tâches par utilisateur est aussi difficile, si on prend par utilisateur la somme des retards car la somme des retards est déjà un problème NP difficile pour le cas mono-utilisateur [Du et Leung, 1990]. [Lai et Kuo, 1996] présente pour ce cas un algorithme de complexité $O(nlog(n))$ avec un ratio de performance au pire de $n/2$.

4.2.5 Critère du débit et du stretch

Comme le débit est l'inverse du ralentissement, les résultats présentés sont valides aussi pour le ralentissement.

Le critère du stretch a été proposé dans [Bender et al., 1998] comme une mesure pour l'équité entre tâche. Ils montrent que le problème $1||S_{\max}$ ne peut être approché par aucun algorithme polynômial par un facteur $O(n^{1-\epsilon})$ pour tout $\epsilon > 0$ (à moins que $P = NP$). [Bender et al., 2002] étudient le problème $1|pmtn, online|S_{\max}$ et proposent un algorithme $O(\Delta^{1/2})$-compétitif, où Δ est le rapport entre la durée maximale et minimale des tâches. [Bender et al., 2004] proposent un PTAS pour le problème $1|pmtn, online|\sum S_i$.

[Legrand et al., 2005] présentent des heuristiques, ainsi qu'un algorithme en ligne basé sur la programmation linéaire pour l'ordonnancement de tâches divisibles, avec pour objectif la somme des stretch ou de minimiser le stretch maximal. [Stillwell et al., 2009, 2010] utilisent la notion de *rendement* qui est l'inverse du stretch et correspond à notre notion de débit. [Casanova et al., 2010] utilisent le critère du stretch pour proposer un ordonnancement équitable des graphes de dépendance entre tâche et proposent différents algorithmes pour ce problème.

4.2.5.a Le problème $1||[\frac{P^h}{C^h_{\max}}]_{\prec_{Lex}}$

Généralisons la notion de débit aux utilisateurs. Notons $P^h/C^h_{\max}$ le débit d'exécution de l'utilisateur h lorsque toutes les tâches sont disponibles à la date 0. De même, nous avons avec le vecteur d'évaluation des débits :

Théorème 4.6 (STPT). *Les politiques STPT (pour Shortest Total Processing Time) sont optimales pour le problème* $1||[\frac{P^h}{C^h_{\max}}]_{\prec_{Lex}}$, *avec* $P^h = \sum_{i \in U_h} p_i$.

Démonstration. Comme pour le Makespan il n'y a pas d'oisiveté sur la machine entre deux tâches car toutes les tâches sont disponibles à la date 0 et regrouper les tâches par utilisateur est dominant.

Échanger deux utilisateurs successifs $h1$ et $h2$ ne change pas les dates d'exécution des autres utilisateurs et par conséquent leurs débits. Soit t la date de début d'exécution de la première

tâche avant $h1$ et $h2$. Exécuter $h1$ suivi de $h2$ donne les débits suivants : $deb_{h1} = P^{h1}/(t + P^{h1})$ et $deb_{h2} = P^{h2}/(t + P^{h1} + P^{h2})$, exécuter $h2$ suivi de $h1$ donne les débits $deb'_{h2} = P^{h2}/(t + P^{h2})$ et $deb'_{h1} = P^{h1}/(t + P^{h1} + P^{h2})$.

On a $deb'_{h1} \leq deb_{h1}$ car $P^{h1}/(t+P^{h1}+P^{h2}) \leq P^{h1}/(t+P^{h1})$, de même pour $h2$ on a $deb_{h2} \leq deb'_{h2}$ car $P^{h2}/(t + P^{h1} + P^{h2}) \leq P^{h2}/(t + P^{h2})$.

Si $P^{h1} \leq P^{h2}$ alors $deb'_{h1} \leq deb_{h2}$, ainsi $\min(deb'_{h1}, deb'_{h2}) \leq deb'_{h1} \leq \min(deb_{h1}, deb_{h2})$ car $deb'_{h1} \leq deb_{h1}$ et $deb'_{h1} \leq deb_{h2}$. D'où, placer $h1$ avant $h2$ est plus équitable en débit si $P^{h1} \leq P^{h2}$.

Par argument d'échanges successifs, nous pouvons en conclure que les politiques STPT sont les politiques les plus équitables en débit.

■

Dans le cas où il existe plusieurs machines avec des tâches disponibles à la date 0 ; l'argument de regroupement des tâches par utilisateur est encore valide car il suffit de l'appliquer successivement sur chaque machine. Ainsi, l'ordre des $C^h_{\max}$ donne un ordre d'exécution pour les tâches des utilisateurs sur chaque machine.

Lorsque les tâches ne sont disponibles qu'à certaines dates (i.e. r_i quelconques), le regroupement des tâches par utilisateur n'est pas dominant. En effet, on ne peut pas avancer une tâche avant sa date de disponibilité, ce qui risque d'engendrer du retard pour les tâches suivantes.

Remarques et propositions pratiques pour le cas en ligne

Si on exécute une tâche de durée p_1 d'un utilisateur et qu'à $\epsilon > 0$ plus tard la seule tâche d'un autre utilisateur arrive avec une durée p_2, alors la date de fin de cette tâche est supérieure à $t + p_1 + p_2$ et son débit est inférieur à $p_2/(\max(p_2, p_1 + p_2 - \epsilon))$. Ainsi, si p_2 peut être aussi petit que l'on veut alors il n'y a alors aucune garantie sur le débit d'un utilisateur en ligne. Remarquons que l'argument est le même avec plusieurs machines ; il suffit de considérer le temps avant qu'une machine ne soit disponible.

Une échappée possible pour pallier à ce problème serait d'autoriser la préemption des tâches. Lorsque la préemption de tâches est interdite, il est préférable de planifier les exécutions à l'avance avec des réservations, dans le cas où des utilisateurs avec des tâches de courtes durées peuvent cotoyer des utilisateurs avec des tâches de longues durées.

Cependant, lorsqu'il n'est pas possible de planifier l'utilisation et que la préemption est impossible, nous proposons plusieurs directions pour éviter cette perte de garantie de service en maintenant la condition de non-préemption :

1. Il faut forcer une borne supérieure pour p. En effet, si p_1 peut être aussi grand que l'utilisateur le souhaite (volontairement ou par erreur à cause d'une boucle infinie), il peut monopoliser l'ensemble des machines et créer un déni de service pour les autres utilisateurs.
2. De même, il faut forcer une borne inférieure pour p. En effet, utiliser le système pour des durées de tâches trop petite n'a pas d'interêt s'il n'y a pas de préemption des tâches.
3. Une autre approche est de réserver au moins une machine par utilisateur. Chaque utilisateur a alors accès à une machine qui lui est dédiée (une machine personnelle à partir de laquelle il peut lancer ses calculs par exemple). Une autre approche similaire serait de découper l'ensemble des machines en plusieurs groupes où chaque groupe n'exécute que les tâches dont la durée est dans un intervalle donné.

4.3 Équité et ordonnancement périodique

Dans cette section et dans la suite nous rechercherons des ordonnancements multi-utilisateurs avec arrivées périodiques des tâches où l'objectif sera de fournir un service équitable aux différents utilisateurs en terme de débits de calcul obtenus.

Nous nous intéressons au problème périodique ; en effet ce modèle permet d'appréhender un cas où les tâches arrivent de façon continue sur un site, tout en conservant une certaine régularité. Le problème avec arrivée périodique des tâches est donc un cas de figure où un nombre infini de tâches est soumis en continu à un ensemble de machines. De plus des tâches périodiques existent bel et bien dans le cas des grilles de calcul comme les tâches de vérification ou les tâches issues d'instruments de mesure à traitement périodique.

Afin de proposer un bon niveau de performance des techniques d'ordonnancement efficaces sont nécessaires. Des décisions d'ordonnancement inadaptées pourraient dégrader l'ensemble des performances ; ceci est particulièrement vrai lorsque la charge du système est élevée car dans ce cas les utilisateurs doivent se partager des ressources, ce qui peut générer des conflits. La plupart des techniques classiques d'ordonnancement sont inefficaces lorsque la charge est importante et doivent prévoir des adaptations [Eager et al., 1986; Kremien et al., 1993; Erdil et Lewis, 2010].

4.4 Aperçu des problèmes et des approches pour l'ordonnancement périodique

Parallélisation automatique de boucles

Ces types de problèmes se rencontrent en particulier dans l'optimisation de compilation de boucles. Un ensemble d'opérations doit s'exécuter indéfiniment et de façon répétitives [Hanen et Munier, 1995] ; une exécution s'appelle une *itération* de la boucle.

La période d'un ordonnancement est l'intervalle de temps entre deux itérations successives. L'objectif de la parallélisation de boucles [Hanen et Munier, 1995; Munier, 1996; Govindarajan et al., 1996] est de trouver un ordonnancement périodique dont la période est minimale compte-tenu des dépendances entre tâches. L'ordonnancement périodique est un cas particulier des ordonnancements $K-$périodiques, où un ensemble de tâches se répètent toutes les K itérations [Van Dongen et al., 1992; Munier, 1996; Fimmel et Müller, 2001]. Lorsque le nombre de processeurs est illimité, ce problème est de complexité polynomiale [Dinechin, 1994; Hanen et Munier, 1995]. Cependant lorsque le nombre de machines est fixé, ce problème devient NP-difficile [Munier, 1996].

[Hanen et Munier, 1995] aborde le problème de l'éxécution d'un ensemble de tâches avec dépendances cycliques sur un ensemble de machines. Les dépendances entre tâches sont modélisées par un graphe cyclique et l'objectif est de trouver une éxécution de temps de cycle minimal. Les problèmes d'ordonnancement périodique auxquels nous faisons référence ici sont différents. Le contexte de l'ordonnancement cyclique est la minimisation d'un temps de cycle lors de l'optimisation des boucles par un compilateur. Notre contexte pour l'ordonnancement périodique est lié à l'exploitation d'un ensemble de machines identiques par des utilisateurs. On ne cherche pas à minimiser le temps de cycle, il est imposé par l'envoi des tâches des utilisateurs. Il est aussi possible, voire fréquent, que certains utilisateurs aient des tâches de durée bien supérieure à la période.

Une approche répandue pour la résolution de ces problèmes est la *programmation linéaire en nombres entiers*. Par exemple une variable entière $x_{i,t}$ indique si la tâche i commence son exécution à la date t [Kum et Sung, 2001; Ito et al., 1998]. D'autres modélisations existent basée sur

une formulation de graphes de flots [Fimmel et Müller, 2001; Fimmel et Müller, 2002]. [Deschinkel et Touati, 2008] propose un formulation linéaire approché d'ordonnancement de graphes de dépendances périodiques sous contraintes de mémoire, dans le contexte du parallélisme d'instructions.

Contrôle temps réel

Dans le domaine du contrôle temps réel les tâches périodiques constituent la majorité des traitements. Une tâche périoque est typiquement générée par des boucles de contrôle, par la surveillance de systèmes, des détecteurs, via l'acquisition de données, etc. Ces tâches sont exécutées périodiquement à cadence régulière et doivent respecter des contraintes spécifiques aux applications, telles que des dates d'échéance. L'objectif est alors de garantir un rythme d'exécution ainsi que le respect des délais.

Ces problèmes utilisent fréquemment des politiques d'ordonnancement à priorités fixes telles que *Earliest Deadline First* [Jeffay et al., 1991] ou *Rate Monotonic* [Liu et Layland, 1973] celle-ci donne une condition suffisante de faisabilité pour n tâches indépendantes sur une machine avec deadlines égales à la période : si l'utilisation du processeur $\sum_i p_i/T_i$ ne dépasse pas $n(\sqrt[n]{2}-1)$, quantité qui converge vers $\ln 2$.

[Lawler et Martel, 1981] traite le problème d'ordonnancement périodique de tâches préemptives sur plusieurs processeurs avec contraintes de dates échues. Cet article montre qu'il existe un ordonnancement préemptif faisable si et seulement si il existe un ordonnancement périodique de période égale au PPCM des périodes des tâches. [Lawler et Martel, 1981] proposent un modèle linéaire pour le résoudre.

[Bar-Noy et al., 1998] étudie un problème d'ordonnancement avec des tâches génériques qui se répétent indéfiniment avec des contraintes de dépendance et d'exclusion entre tâches. L'objectif est d'allouer des fréquences d'occurrence des tâches en sorte de maximiser une mesure d'équité de type max-min et une mesure de régularité.

4.4.1 Contexte

Contrairement à d'autres articles sur les ordonnancements périodiques, nous nous plaçons du point de vue d'un *site* accueillant plusieurs communautés d'utilisateurs ; lesquelles se partagent les ressources fournies par le site. Le devoir d'un site est de traiter au mieux les utilisateurs et leur flot régulier de tâches. Les décisions concernant l'envoi les tâches sont prises par les utilisateurs. Il y a séparation entre site et utilisateurs ; le site n'a pas d'influence sur la quantité de tâches reçues ou sur leur date d'envoi. Les tâches reçues par le site peuvent être exécutée dans l'ordre souhaité par le site, il n'y a pas de dépendances entre tâches. En revanche, les tâches que nous considérons utilisent un volume mémoire important et ne sont donc pas préemptives pour des raisons d'efficacité (dues à l'utilisation mémoire et réseau) : on ne s'autorise pas à arrêter une tâche en cours d'exécution et à la reprendre plus tard.

Dans la mesure où ce sont les utilisateurs qui décident de l'envoi de leur tâches et non le site qui les reçoit, les modèles avec graphes de dépendances et ayant pour objectif de minimiser le temps de cycle ne sont pas adaptés à notre cas d'étude.

Un ensemble d'utilisateurs (ou de clients) se partagent l'utilisation d'un site qui contient un ensemble de machines. Chaque utilisateur a un ensemble de tâches périodiques à exécuter (acquisition et traitement de données, tâches de vérification et de surveillance). Chacune de ces tâches

est disponible périodiquement, la période ne dépend que de la tâche générique considérée. Une tâche générique possède une durée et appartient à un utilisateur particulier.

Notons qu'une part importante des tâches sur une grille de calcul sont des tâches de longue durée : celle-ci dépasse largement la période de soumission. Il n'y a pas de contraintes de terminer une tâche avant la période suivante.

4.4.2 Données du problème et leurs notations

Nous allons présenter les données de notre problème et leurs notations.

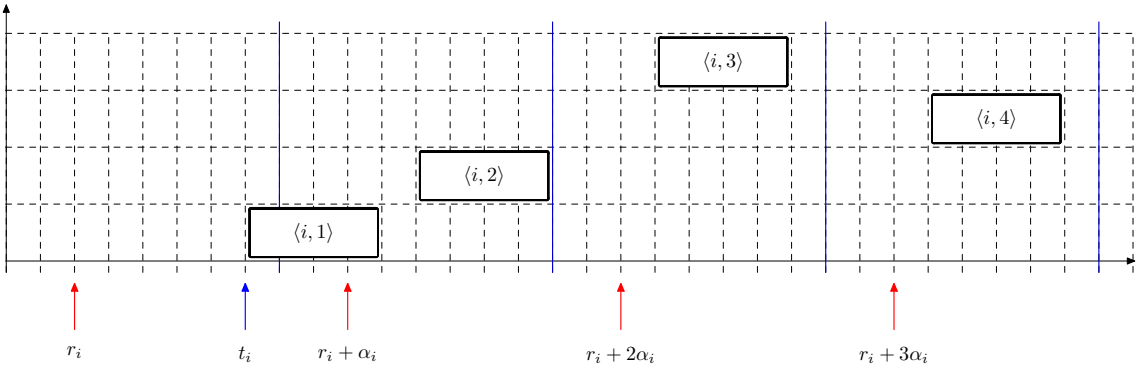


Fig. 4.4.1 Les tâches i arrivent avec une périodicité α_i.

Définition 4.7 (Tâche générique [Hanen et Munier, 1995])**.** *Une tâche générique est un ensemble de tâches qui appartiennent à un utilisateur donné et ont chacune des caractéristiques identiques.*

Les tâches considérées ici sont périodiques et ont chacune même durée. On considère connues les données suivantes :

Caractéristiques des utilisateurs

Nombre d'utilisateurs : N

Indices des utilisateurs : Les utilisateurs sont indicés par $h \in \{1, \ldots, N\}$. On parle aussi de l'utilisateur h.

On associe à chaque utilisateur son ensemble de tâches. L'ensembe des tâches de l'utilisateur h est noté U_h. Il arrive de confondre l'utilisateur h avec l'ensemble U_h de ses tâches lorsqu'il n'y a pas d'ambiguïtés.

Nombre total de tâches génériques à ordonnancer pour un utilisateur : U_h possède N_h^{op} tâches génériques.

Afin de ne pas surcharger les notations, on utilise un seul indice pour énumérer les tâches de tous les utilisateurs et non un double indice contenant celui de la tâche et celui de l'utilisateur.

Caractéristiques des tâches génériques

Indices des tâches génériques : i

Indice d'occurrence des tâches : $k \in \mathbb{N}^*$. Les tâches sont numérotées selon leur ordre d'arrivée.

Occurrences d'une tâche générique : la k^e occurrence de la tâche générique i est notée $\langle i, k \rangle$.

Nombre total de tâches génériques à ordonnancer N^{op}

Une tâche générique i possède :
- une date de première occurrence $r_i = r_{\langle i,1 \rangle}$
- une quantité de travail p_i égale à sa durée lorsque les machines sont identiques.
- une périodicité α_i, de sorte que la date de disponibilité de la tâche $\langle i, k+1 \rangle$ est $r_{\langle i,k+1 \rangle} = r_i + k\alpha_i$.

Caractéristiques des machines

Les machines (ou serveurs) sont supposées identiques.

Nombre de machines : M

Indices des machines : Les machines sont numérotées avec $l \in \{1, \dots M\}$.

Nous pouvons désormais définir la notion d'ordonnancement de tâches périodiques sur des machines identiques :

Définition 4.8 (Ordonnancement de tâches périodiques)**.** *Un ordonnancement de tâches périodiques est la donnée pour chaque tâche* $\langle i, k \rangle$ *de sa date d'exécution* $t_{i,k}$.

Définition 4.9 (Réalisation d'un ordonnancement périodique)**.** *La réalisation d'un ordonnancement contient en plus des caractéristiques de chaque tâche, la machine sur laquelle cette tâche est exécutée. Sachant qu'une machine ne peut exécuter qu'une seule tâche à un instant donné et que les tâches ne peuvent pas être interrompues (pas de préemption des tâches).*

Notons bien que nous n'imposons pas dans cette définition que l'*exécution* des tâches soit périodique. En effet, nous pouvons décider de traiter ces tâches de façon périodique ou ne pas s'imposer une exécution périodique. Cependant pour des raisons pratiques, nous rechercherons, parmi tous les ordonnancements possibles, des ordonnancements *dont les dates d'exécutions sont périodiques* car de tels ordonnancements sont simples à implémenter. Nous avons donc séparé la notion d'arrivée périodique des tâches de la notion de traitement périodique. La question de savoir si selon les critères étudiés les ordonnancements à exécution périodique sont ou non dominants reste ouverte : existe-t-il une façon d'éxécuter les tâches qui ne soit pas périodique et qui maximise le critère donné ?

Définition 4.10 (Union d'ordonnancements)**.** *Soient les ordonnancements* O *et* O' *composés de tâches différentes. L'union* $O \cup O'$ *est un ordonnancement constitué des tâches de* O *et des tâches de* O'*, de sorte que chaque tâche commence son exécution à sa date de début d'exécution dans* O *ou* O'.

On dit aussi que $O \cup O'$ étend O avec O' ou que $O \cup O'$ est une extension de O. La notion d'union d'ordonnancements permet de décomposer un ordonnancement en sous-ordonnancements et sera utilisée par la suite.

Caractéristiques des ordonnancements

Périodicité globale d'exécution de l'ordonnancement recherché : T

T désigne la période d'exécution. Notons que d'après le lemme 4.8 T doit être un multiple des α_i, ainsi tout multiple du PPCM des α_i convient.

Nombre de périodes considérées : même si le nombre de période est en principe infini, nous nous bornons en pratique à l'étude de Γ périodes consécutives.

Nombre d'occurrences de la tâche i : aprés la date de première disponibilité d'une tâche générique i il arrive pendant un intervalle de temps T, $K_i = T/\alpha_i$ tâches à traiter.

Eventuelles contraintes de précédences imposées : on suppose connu un ensemble de contraintes supplémentaires entre tâches. Une telle contrainte s'écrit $\langle i_1, k_1\rangle \rightarrow \langle i_2, k_2\rangle$ ce qui signifie que la tâche $\langle i_2, k_2\rangle$ débute après la date de fin de la tâche $\langle i_1, k_1\rangle$. De plus, cette contrainte se répéte périodiquement : on doit avoir aussi $\forall a \in \mathbb{N}, \langle i_1, k_1 + aK_{i_1}\rangle \rightarrow \langle i_2, k_2 + aK_{i_2}\rangle$. Dans les modèles présentés nous pouvons, si besoin, ajouter un temps d'attente avant le début de $\langle i_2, k_2\rangle$. Ces contraintes sont optionnelles.

Définition 4.11 (Ordonnancement périodique à une permutation des machines près)**.** *Un ordonnancement est dit* périodique à une permutation des machines près *si et seulement s'il existe une permutation des machines σ telle que l'ordonnancement $\mathcal{S}$ soit une succession d'ordonnancements $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$ et telle qu'on obtient $\mathcal{S}_{a+1}$ pour $a \geq 2$ à partir de $\mathcal{S}_a$ en permutant les machines selon σ et en décalant d'une période. (Voir la figure 4.4.2)*

Dans la décomposition $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$:

- *La partie $\mathcal{S}_0$ est appelée* phase (ou régime) transitoire.
- *La partie $\mathcal{S}_1$ est appelée* motif.
- *La partie $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots$ est appelée* régime permanent.

Notons que cette décomposition n'est pas forcément unique.

Dans le cas périodique, nous considérons seulement les ordonnancements constitués d'un motif qui se répéte périodiquement et d'une phase transitoire avant l'apparition du motif. En effet, parmi les ordonnancements possibles certains n'exécutent pas régulièrement les tâches.

Exemple d'ordonnancement périodique à permutation de machines près

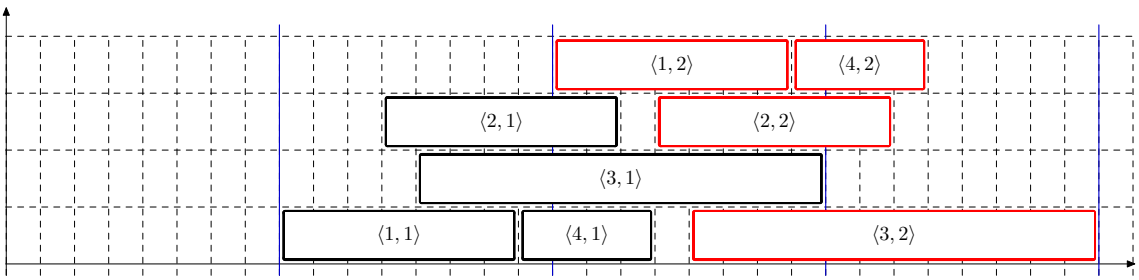


Fig. 4.4.2 Ordonnancement périodique à une permutation de machines près.

Soient les tâches génériques suivantes de même période 8 :

$r_1 = 0$ $p_1 = 7$
$r_2 = 3$ $p_2 = 7$
$r_3 = 4$ $p_3 = 12$
$r_4 = 7$ $p_4 = 4$

La figure 4.4.2 montre une exécution possible avec un motif $\mathcal{S}_1$ en noir et la période suivante $\mathcal{S}_2$ en rouge. Sur l'exemple de la figure 4.4.2, nous avons 4 machines, seuls $\mathcal{S}_1$ en noir et $\mathcal{S}_2$ en rouge sont représentés. La permutation de machines est $\left(\begin{smallmatrix}1&2&3&4\\4&1&3&2\end{smallmatrix}\right)$. En effet, la première machine reprend l'ordonnancement de la machine 4 après une période, etc.

Autre exemple

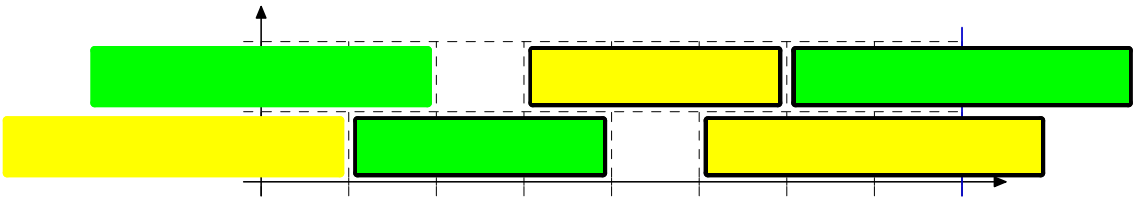


Fig. 4.4.3 Motif d'un ordonnancement périodique. (tâches avec bordure épaisse)

L'exemple suivant de la figure 4.4.3 montre un motif d'un ordonnancement périodique avec 2 machines et 2 utilisateurs. Les tâches sont toutes de période 8. Les caractéristiques des tâches sont les suivantes :

i	r_i	p_i	U_i
1	5	4	Jaune
2	6	4	Vert
3	3	3	Jaune
4	1	3	Vert

Tab. 4.1 Exemples de tâches

Les tâches sans bordure correspondent à des tâches de la période précédente. La permutation de machines est l'identité. On remarquera sur cet exemple que la permutation est composée de 2 cycles qui sont $(1)(2)$ et que toutes les tâches commencent leur exécution à leur date d'arrivée.

On parle aussi d'ordonnancement à éxécution périodique pour insister sur le fait que les arrivées peuvent être périodiques sans que les exécutions le soient.

Dans le cas d'ordonnancement périodique à une permutation des machines près, c'est la même permutation qui réplique l'ordonnancement sur la période d'arrivée suivante et ainsi de suite. Remarquons qu'il existe alors un entier a tel que $\sigma^a = id$. Ainsi on retrouve l'ordonnancement $\mathcal{S}_1$ après a périodes.

Considérons la décomposition de la permutation σ en produit de cycles à support disjoint. Soit $(m_1, \ldots, m_a)$ un cycle de ce produit. Cela implique que chaque machine $m_1, \ldots, m_a$ exécute la même suite de tâches avec un décalage dans les périodes. Sur l'exemple les machines $\{1, 2, 4\}$ exécutent une tâche de l'ensemble des tâches génériques : 1 suivie de 4 et de 3 et recommence le

cycle. La machine 3 exécute successivement les tâches de la tâche générique 2. Ainsi, la machine 1 exécute la suite de tâches

$$\langle 1,1\rangle, \langle 4,1\rangle, \langle 3,2\rangle, \langle 1,3\rangle, \langle 4,3\rangle, \langle 3,4\rangle, etc.$$

et les machines du même cycle sont les machines 2 et 4 ayant des suites identiques à un décalage près.

Chaque tâche est périodique de période α_i, mais si on considère globalement l'ensemble des tâches et leurs dates d'arrivées on peut envisager une période globale d'arrivée qui est le temps après lequel l'ensemble des tâches se répéte. Remarquons que la période globale d'arrivée des tâches n'est pas forcément la période d'exécution de l'ordonnancement. Nous verrons qu'il peut en être un multiple. On parle aussi de temps de cycle :

Définition 4.12 (Temps de cycle)**.** *Le temps de cycle T^c d'un ordonnancement à arrivées périodiques est la période globale des arrivées des tâches. C'est le* PPCM *des périodes de chaque tâche [Brakerski et al., 2002].*

Le temps de cycle peut être différent des périodes d'arrivée des tâches, simplement parce que les tâches génériques peuvent avoir des périodes distinctes. Le temps de cycle est aussi distinct de la période de l'ordonnancement. Nous verrons dans le lemme 4.8 que la période est un multiple du temps de cycle.

Définition 4.13 (Bande passante et Quantité de calcul)**.** *Le débit demandé [Brakerski et al., 2002] par une tâche générique i est le rapport $\frac{p_i}{\alpha_i}$. On trouve aussi les termes de bande passante ou d'utilisation [Liu et Layland, 1973] pour cette quantité.*

Le débit d'entrée d'un utilisateur h est la somme des débits demandés par ses tâches.

La quantité de calcul traitée pendant un motif de durée T est notée Q_h pour un utilisateur U_h. C'est aussi la somme des quantités de calcul reçues pendant une période de la part de l'utilisateur U_h.

Le rapport Q_h/T est appelé bande passante offerte à l'utilisateur U_h. Si cette quantité converge pour un ordonnancement quelconque donné, on dit que l'ordonnancement possède un débit stable *[Jagabathula et Shah, 2008].*

La bande passante libre [Brakerski et al., 2002] est la quantité $M - \sum_{i=1} \frac{p_i}{\alpha_i}$.

Reprenons l'exemple ; ainsi la bande passante demandée par la tâche générique 3 est de $12/8$, la bande passante libre est de $4 - (7 + 7 + 12 + 4)/8 = 1/4$

Remarque sur le motif

Ce qui constitue un motif n'est pas seulement le nombre d'occurrences de chaque tâche (du fait des parties découpées par le motif) mais bien la quantité de calcul présente pendant la période.

Si on considère la seconde période dans l'exemple de la figure 4.4.4, le nombre d'occurrences de la tâche jaune est de 1, le nombre d'occurrence de la tâche verte est de 2 et le nombre d'occurrences de la tâche bleu est de 1. Pourtant cette période ne contient pas un motif complet car elle ne contient pas la fin de la tâche jaune tout comme la période précédente n'en contenait pas.

Le motif apparait complet dans la troisième période et on a aussi les mêmes nombres d'occurrences de chaque tâches. Chaque tâche générique i doit consommer exactement $p_i T/\alpha_i$ unités de ressources dans un motif.

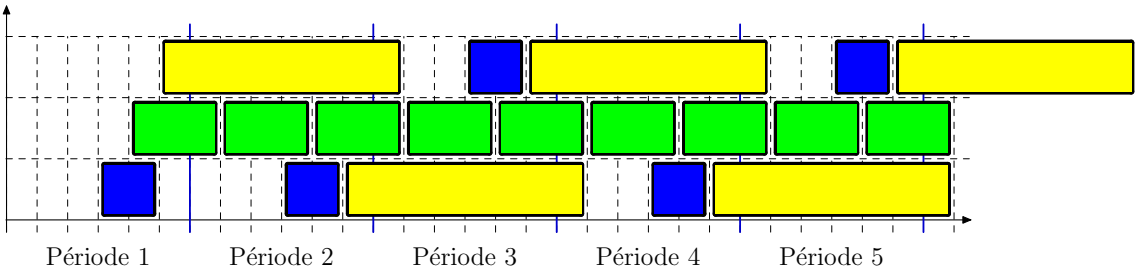


Fig. 4.4.4 Exemple de motif incomplet

Donc, pour la tâche jaune, $p_i = 8, \alpha_i = T = 6$, la quantité de ressources utilisées est de 8.

Pour la tâche verte on a $p_i = 3, \alpha_i = 3, T = 6$, la quantité de ressources utilisées est de 6.

Si on regarde la période 2 du dessin 4.4.4, on constate que la tâche jaune consomme 7 et que la tâche verte consomme 6, donc comme $7 \neq 8$, on sait que le motif ne contient pas tous les morceaux de tâches.

Observons que ceci n'a de sens que parce que les machines sont identiques. Dans le cas contraire, il faut considérer non pas une durée par tâche, mais une quantité de calcul par tâche et avoir des vitesses par machine. De plus, l'exécution périodique avec des machines hétérogènes n'a pas beaucoup de sens car les durées des tâches dépendent des machines. Cela implique qu'en traversant une période une tâche reste sur une machine de vitesse identique, ce qui restreint les permutations de machines possibles.

4.4.3 Réalisabilité d'un ordonnancement à arrivées périodiques

Définition 4.14 (Réalisabilité d'un ordonnancement à arrivées périodique)**.** *Soit $QNT(x)$ la quantité de calcul non traitée au temps x. Un problème d'ordonnancement à arrivées périodique est dit réalisable si et seulement s'il existe une façon d'ordonnancer les tâches en sorte que QNT ne diverge pas, c'est à dire qu'elle soit bornée (on dit aussi que l'ordonnancement est* admissible *[Jagabathula et Shah, 2008]).*

Remarquons que si un problème d'ordonnancement périodique est réalisable alors, la quantité de calcul non traitée (ou le nombre de tâches non traitées) par utilisateur est bornée aussi.

Nous avons la condition de réalisabilité suivante :

Théorème 4.7 (Condition de réalisabilité d'un ordonnancement périodique)**.** *Un problème d'ordonnancement périodique est réalisable si et seulement si :*

$$\sum_{h=1}^{N} \sum_{j=1}^{n_h} \frac{p_{j,h}}{\alpha_{j,h}} \leq M$$

Démonstration. Si $\sum_{h=1}^{N} \sum_{j=1}^{n_h} \frac{p_{j,h}}{\alpha_{j,h}} > M$, alors cela signifie que par période on reçoit plus de quantité de calcul à traiter que les machines sont capables de traiter à chaque période. Ainsi dans ce cas la quantité de calcul à traiter diverge.

Démontrons que si $\sum_{h=1}^{N} \sum_{j=1}^{n_h} \frac{p_{j,h}}{\alpha_{j,h}} \leq M$ alors le problème d'ordonnancement périodique est réalisable. La preuve de réalisabilité utilise un ordonnancement similaire à la solution fournie par

la règle de McNaughton [McNaughton, 1959] pour le problème du Makespan avec moins de m préemptions.

Soit T le PPCM des $\alpha_{j,h}$ et $d_{\max} = \max d_{j,h}(0)$.

Si on attend jusqu'à $t = T + d_{\max}$ sans traiter de tâches, on a reçu au moins une tâche de chaque type. Donc, la quantité de calcul non traitée par utilisateur à l'instant t (notée $QNT_h(t)$) est alors au moins égale à la quantité de calcul reçue entre $d_{\max}$ et $T + d_{\max}$ ce qui est $\sum_{j=1}^{n_h} p_{j,h} \frac{T}{\alpha_{j,h}}$. Ainsi, $QNT_h(T + d_{\max}) \geq \sum_{j=1}^{n_h} p_{j,h} \frac{T}{\alpha_{j,h}}$. Notons Q_h la quantité de calcul reçue par période de durée T, à partir de $T + d_{\max}$ cette quantité est constante pour chaque utilisateur et on a $Q_h = \sum_{j=1}^{n_h} p_{j,h} \frac{T}{\alpha_{j,h}}$.

Etant donné que $Q_h(T + d_{\max}) \geq Q_h$, nous avons à disposition pour chaque utilisateur U_h au moins une quantité de calcul Q_h. Pendant chaque période de durée T après une certaine date, s'il est possible d'exécuter pour l'utilisateur U_h une quantité de calcul Q_h la quantité de calcul non traitée restera bornée. Montrons que cela est possible.

Soit sur une machine l'ordonnancement qui consiste à exécuter d'abord Q_1 de l'utilisateur U_1, ensuite Q_2 de l'utilisateur U_2 et ainsi de suite. Une fois la quantité de calcul Q_N traitée ajoutons de l'oisiveté jusqu'au temps mT, ceci est toujours le cas puisque par hypothèse $\sum_{h=1}^{N} \sum_{j=1}^{n_h} \frac{p_{j,h}}{\alpha_{j,h}} \leq M$ donc $\sum_{h=1}^{N} \sum_{j=1}^{n_h} p_{j,h} \frac{T}{\alpha_{j,h}} = \sum_{h=1}^{N} Q_h \leq MT$. Ensuite, on recommence périodiquement l'ordonnancement obtenu sur la période T.

Il suffit alors de replier l'ordonnancement obtenu pour une machine sur m machines, c'est à dire que la machine $i = 0 \ldots M-1$ exécute de la même façon ce que fait l'ordonnancement précédent sur une seule machine mais avec un retard de iT. Ceci est rendu possible car il y a assez de quantité de calcul disponible au début et à chaque période, on consomme ce qui est reçu par période c'est à dire $\sum_{h=1}^{N} Q_h \leq MT$.

Le problème est réalisable puisque après la date $MT + d_{\max}$, on consomme pour chaque utilisateur la quantité de calcul qu'on reçoit de sa part pendant la période ; donc la quantité de calcul non traitée reste bornée.

■

Notons que l'ordonnancement obtenu dans la preuve de réalisabilité est un ordonnancement périodique et que nous n'avons pas imposé la périodicité de l'ordonnancement dans la définition de la réalisabilité (les tâches seules sont supposées arriver périodiquement). Par conséquent, si le problème est réalisable il existe un ordonnancement périodique.

Exemples

Reprenons l'exemple de la figure 4.4.2, comme $(7+7+12+4)/8 = 3.75 \leq 4$, l'ordonnancement est réalisable. Si maintenant on modifie la durée de la tâche 1 en 10 unités de temps, l'ordonnancement devient infaisable car $33/8 > 4$.

Remarque 1. *Lorsqu'on ne considère que les tâches d'un seul utilisateur U_h, l'ordonnancement obtenu reste réalisable et utilise M_h machines, en ne tenant pas compte des machines non utilisées par cet utilisateur.*

Si p_i désigne la durée d'une tâche et α_i sa période, on a donc aussi pour chaque U_h :

$$\sum_{i \in U_h} p_i/\alpha_i \leq M_h$$

i.e. le nombre de machines m_h qu'utilise U_h est au moins égal à son débit d'entrée $\sum_{i \in U_h} p_i/\alpha_i$. Etant donné que le nombre de machines est un entier, on doit avoir si $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x : $M_h \geq \lceil \sum_{i \in U_h} p_i/\alpha_i \rceil$.

Considérons le cas où chaque tâche possède sa propre période. Alors, on peut construire un problème équivalent où chaque tâche a une période qui est un multiple du PPCM des périodes de toutes les tâches : il suffit de répéter un type de tâche autant de fois que nécessaire avec des dates d'arrivée espacée de la période de cette tâche.

Soit une instance d'un problème à arrivées périodiques, alors on peut remplacer chaque tâche générique i par T^c/α_i tâches génériques de durée identique avec la date de première occurrence : $a = 1 \ldots T^c/\alpha_i, r_{i_a} = r_i + (a-1)\alpha_i$ et de périodicité T^c. En appliquant cela à toutes les tâches génériques on obtient un nouveau problème équivalent où toutes les tâches génériques ont la même période T^c. Appelons ce problème équivalent le *transformé* du problème initial.

Lemme 4.8. *Soit un ordonnancement périodique à une permutation des machines près alors la période d'exécution T doit être un multiple de la période globale d'arrivée T^c qui est le PPCM des périodes d'arrivées de chaque tâche.*

Démonstration. Prenons le transformé du problème, ainsi on a avec une période globale identique pour toutes les tâches. On reçoit pendant T $\lfloor T/T^c \rfloor$ ou $\lceil T/T^c \rceil$ tâches. La partie de l'ordonnancement éxécuté pendant T se répéte. On exécute pendant T le même nombre de tâches donc si T n'est pas un multiple de T^c et si on exécute $\lfloor T/T^c \rfloor$ tâches le nombre de tâches divergerait, et si on exécute $\lceil T/T^c \rceil$ on n'en reçoit pas assez pour combler chaque période. Donc T est un multiple de T^c. ∎

Notons que si on note K tel que $T = KT^c$, cela rejoint la notion d'ordonnancements K-périodiques [Munier, 1996].

Un ordonnancement de période d'exécution T (multiple de T^c) est aussi de période $\forall k \in \mathbb{N}, kT$. Ainsi les solutions des problèmes d'optimisation où la période est fixée à T sont aussi solutions des problèmes d'optimisation avec pour période kT. Par conséquent les solutions optimales pour kT sont au moins aussi bonnes que les solutions optimales pour T.

4.4.4 Étude de la phase transitoire

Un ordonnancement périodique (Voir figure 4.4.5) se décompose en une première phase transitoire, suivie d'une phase périodique, constituée de la répétition d'un motif. La première période où toutes les tâches sont disponibles est appelée "période de complète disponibilité".

Nous considérerons seulement les ordonnancements constitués d'un motif se répétant périodiquement avec une phase transitoire avant l'apparition du motif.

Supposons qu'on se fixe un motif à réaliser alors, on peut construire un ordonnancement périodique avec ce motif et une phase transitoire de durée bornée :

Théorème 4.9 (Construction d'une phase transitoire)**.** *Il est possible de construire la phase transitoire de sorte que le motif soit complet en au plus $2 + M$ périodes aprés la période où toutes les tâches génériques sont disponibles. Si toutes les durées des tâches sont inférieures à la période alors, on peut construire une phase transitoire de sorte que le motif soit complet au plus tard à la 3e période.*

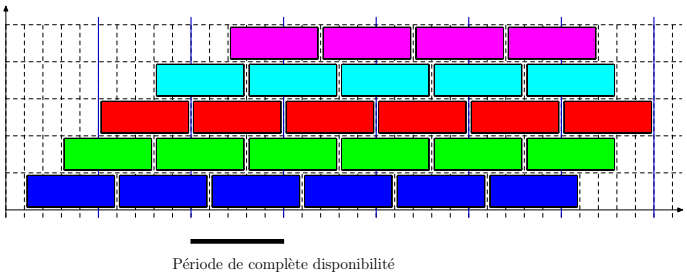


Fig. 4.4.5 Représentation d'un ordonnancement périodique et période de complète disponibilité

Démonstration. L'idée est de remplir la partie transitoire avec des motifs incomplets. Pour cela considérons la suite croissante $r_1, \ldots$ des dates d'arrivées pour une tâche générique donnée et la suite croissante des dates de début d'exécution $t_1, \ldots$ imposées par le motif.

L'objectif est de faire correspondre à chaque tâche une date de début d'exécution dans le motif. Il faut donc trouver le bon décalage de sorte que les dates de début d'exécution soient valides : $\forall i, r_i \leq t_{i+\delta}$.

Soit δ_i le décalage minimal pour que $r_i \leq t_{i+\delta_i}$, ce décalage existe car $t_{i+K} = t_i + T$.

Soit $\delta = \max_{i=1,\ldots,K} \delta_i$, on a alors $\forall i, r_i \leq t_{i+\delta}$.

Notons que tout décalage $\delta' > \delta$ est aussi valide, mais il induit un retard plus grand sur les tâches.

Calculons le nombre maximum de périodes transitoires, comme :

$$r^1 \leq t^1 \leq r^1 + T$$

On en déduit que la dernière tâche se termine à $t^{K_i} + p_i$. Or on a $t^{K_i} + p_i \leq 2T + p_i$. Donc le motif est complet à la période $\lceil 2 + \frac{\max p_i}{T} \rceil$.

Comme $\sum p_i/\alpha i \leq M$, on a $\forall i, p_i \leq MT$. Ainsi, les motifs sont complets en au plus $2+M$ périodes aprés la phase où toutes les tâches sont présentes.

Cas particulier : si $\forall i, p_i \leq T$ le motif est complet au plus tard à la 3e période. Par exemple pour la figure suivante :

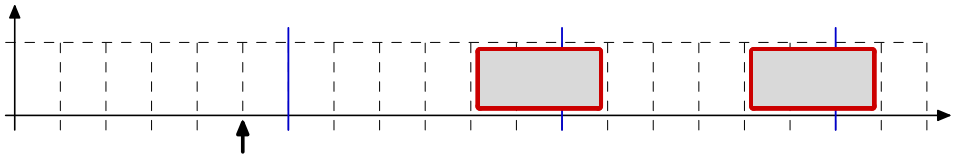


Fig. 4.4.6 Motif complet à la 3e période.

■

Ainsi si les dates d'exécution dans le motif sont fixées, il faut trouver un décalage des indices afin que les dates de disponibilités soient toutes inférieures à la date de début d'exécution pour chaque tâche. Les figures 4.4.7 et 4.4.8 illustrent cette recherche. Observons qu'une tâche qui a dans le motif une date d'exécution avant sa date de disponibilité est en fait une tâche qui a été soumise à la période précédente et qui s'exécute pendant la période dans le motif.

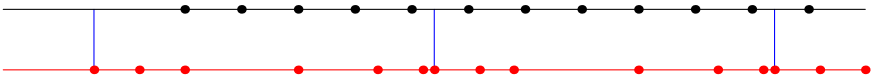


Fig. 4.4.7 Suite des dates de disponibilités (en noir) et suite des dates d'exécution (en rouge). Les périodes sont dessinées en bleu.

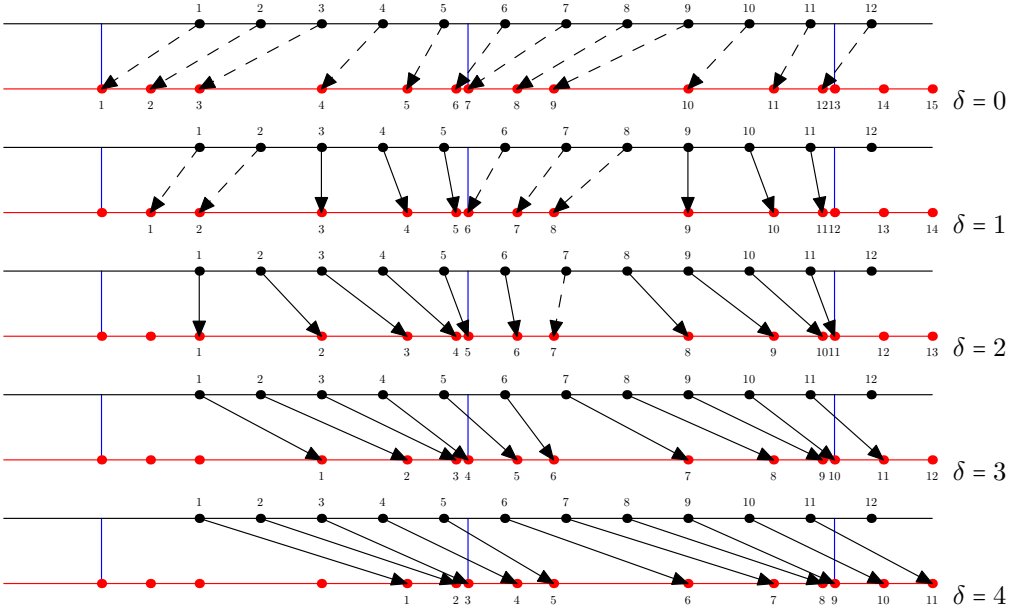


Fig. 4.4.8 Dates d'arrivées des tâches et dates d'exécution imposées par le motif. Une flèche en pointillée indique que la correspondance entre date de disponibilité et date d'exécution est invalide.

4.5 Dominances

Parmi les ordonnancements possibles, nous rechercherons les meilleurs selon un critère donné.

Définition 4.15 (Critère régulier)**.** *Un critère est régulier si la qualité d'un ordonnancement se dégrade lorsqu'on retarde l'exécution des tâches.*

Nous supposerons désormais que le critère est régulier et qu'il ne dépend pas de la phase transitoire. Donnons alors un ensemble de propriétés de dominances.

Théorème 4.10 (Borne sur le temps d'exécution d'une tâche)**.**

$$r_{i,k} \leq t_{i,k} \leq r_{i,k} + \alpha_i$$

Démonstration. Soit un ordonnancement périodique quelconque, considèrons l'ordonnancement de même motif, dont la partie transitoire a été construite à partir de motifs partiels comme ci-dessus. Etant donné qu'on peut reconstruire un ordonnancement périodique à partir d'un motif donné et que le critère ne dépend pas de la phase transitoire, nous allons modifier le motif puis

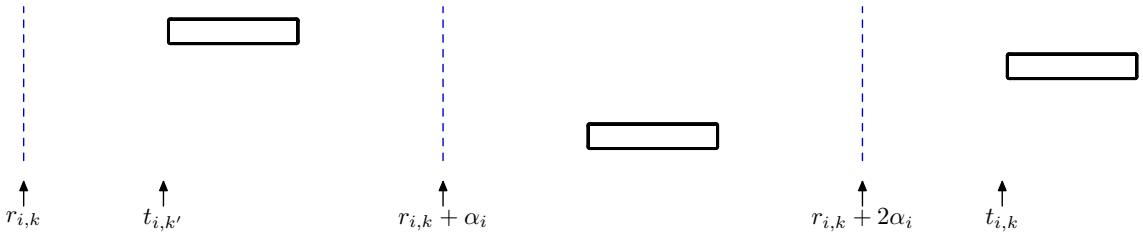


Fig. 4.5.1 Décalage des temps d'exécution dans une période

au besoin, reconstruire une phase transitoire (par exemple composée de motifs partiels comme ci-dessus).

Si $t_i > r_i + T$ alors, étant donné que l'exécution est périodique [1], on peut décaler toutes les exécutions de tâches d'un cran : soit t_i en $t'_i = t_i - T$. Cette transformation améliore la qualité de l'ordonnancement car le critère est régulier.

Ainsi en répetant le décalage, chaque tâche s'exécute soit dans la période de sa date de disponibilité, soit dans la période suivante car comme $r_i \in [0, T]$, on a $t_i \in [0, 2T]$. ■

Démontrons maintenant que les ordonnancements semi-actifs [Brucker, 2001] sont dominants pour notre problème :

Théorème 4.11 (Dominance des ordonnancements semi-actifs)**.**

$$t_{i,k} = r_{i,k} \text{ ou } \exists \langle i', k' \rangle \, t_{i,k} = t_{i',k'} + p_{i'}$$

Démonstration. Comme le critère est supposé régulier, il se détériore lorsque les $t_{i,k}$ augmentent tous. Ainsi, il faut exécuter une tâche s'il y a des tâches disponibles et si les machines sont libres. En effet, s'il existe un temps d'oisiveté sur une machine avant l'exécution d'une tâche et que cette tâche n'est pas calée à sa date de disponibilité, on peut améliorer l'ordonnancement en avançant cette tâche. Par conséquent, chaque tâche est soit calée sur sa date de disponibilité soit calée sur la fin d'une tâche précédente.

Pour des raisons similaires (en transposant les tâches), les tâches d'un même type peuvent s'exécuter dans l'ordre de leur arrivée.

■

Algorithme de calcul d'un ordonnancement calé à gauche dans le cas périodique à une machine

Le cas particulier d'une machine avec tâches calées périodiquement à gauche souligne la particularité des ordonnancements périodiques. En effet, caler les tâches à gauche n'est pas immédiat puisque les tâches qui s'exécutent à cheval sur une période future peuvent décaler les tâches de la période actuelle. Il n'y a donc pas de notion de "première" tâche.

Donnons la méthode de calcul d'un ordonnancement calé à gauche dans le cas périodique à une machine : on a un ensemble de N de tâches (r_i, p_i) de période T, triées par ordre de disponibilité croissante.

1. Le critère ne dépend pas de la phase transitoire on peut donc considérer que la phase transitoire est calculée comme un motif incomplet

Calculons la date de début d'exécution de la première tâche dans la période suivante :

$$\begin{aligned} T + t_1 &= \max(T + r_1, \quad t_N + p_N) \\ &= \max(T + r_1, \quad r_N + p_N, t_{N-1} + p_N + p_{N-1}) \\ &= \dots \\ &= \max(T + r_1, \quad r_N + p_N, \\ &\qquad\qquad r_{N-1} + p_N + p_{N-1}, \\ &\qquad\qquad \dots \\ &\qquad\qquad r_1 + p_N + \dots + p_1, \\ &\qquad\qquad t_1 + p_N + \dots + p_1) \end{aligned}$$

Si $p_N + \dots + p_1 = T$ alors toute date t est solution pour t_1 et la machine est occupée dans tout l'intervalle $[0, T]$.

Sinon comme $T + t_1 > t_1 + p_N + \dots + p_1$ on a

$$t_1 = \max(T + r_1, r_N + p_N, r_{N-1} + p_N + p_{N-1}, \dots, r_2 + p_N + \dots + p_2) - T$$

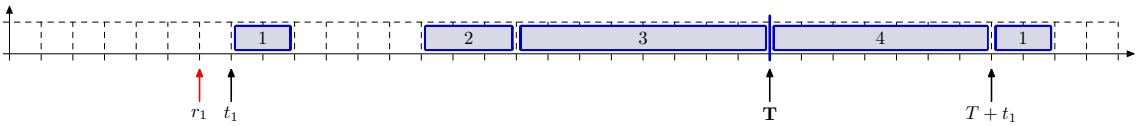


Fig. 4.5.2 Cas de figure sur une seule machine avec tâches calées à gauche. $T = 24, r_1 = 6, p_1 = 2, r_2 = 13, p_2 = 3, r_3 = 15, p_3 = 8, r_4 = 19, p_4 = 7$. On voit sur cet exemple que $t_1 \neq r_1$.

Ce résultat sur une seule machine est aussi utilisable avec plusieurs machines lorsque chaque tâche est mono-processeur [2] : En effet, à cause des cycles de la permutation chacune des machines exécute périodiquement un sous-ensemble de tâches en les calant à gauche, ainsi cet algorithme quoi que simple mérite d'être remarqué. Soulignons encore la signification de "tâches calée à gauche" dans le cas périodique : si la somme des durées des tâches est strictement inférieure à la période alors, une des tâches est calée sur sa disponibilité et non pas forcément la première. Si la somme des durées est égale à la période alors chaque tâche s'exécute à la suite.

2. Comme c'est actuellement le cas dans le contexte des grilles de calculs comme WLCG.

4.6 Définition du principe de comparaison

Rappelons ici qu'un critère est tout d'abord vu comme un *principe de comparaison entre utilisateurs*. Le critère choisi va donc nous permettre de comparer les parts reçues par chaque utilisateur. Le critère de comparaison ne doit dépendre pour chaque utilisateur que des tâches de cet utilisateur et non des tâches des autres utilisateurs.

4.6.1 Justification du critère

Il nous reste à définir quelle est la métrique de performance par utilisateur, que l'on cherche à rendre équitable. On souhaite un critère de comparaison entre utilisateurs qui tienne compte de l'utilisation des machines. Prenons un exemple avec une machine : soit une tâche qui consiste à exécuter une commande sur F fichiers successivement de façon indépendante. Soit aussi F tâches exécutées à la suite l'une de l'autre où chacune exécute la commande sur chaque fichier. Il nous semble raisonnable de dire que ces deux exécutions sont identiques du point de vue de la performance et donc, doivent avoir la même valeur lors de la comparaison. Si un utilisateur a le premier ordonnancement et un autre le second aucun ne sera considéré comme désavantagé. Certaines métriques, comme la somme des temps de complétion des tâches, ne vérifient pas cette propriété et sont sensibles au découpage en sous-tâches. Ceci revient à dire que la métrique recherchée ne doit dépendre que des dates d'arrivées des tâches ainsi que du débit instantanné reçu en fonction du temps.

[Baker et Smith, 2003] considérent un critère qui est une combinaison linéaire des objectifs des différents utilisateurs. Ils montrent que les utilisateurs ont alors interêt à regrouper leurs tâches en une seule afin d'optimiser ce critère. Pour éviter ce phénomène, nous prenons le principe suivant : le fait de remplacer une tâche par une succession de sous-tâches de même durée totale ne doit pas modifier le critère. En effet, une tâche peut être constituée elle-même de multiples sous-tâches.

Définition 4.16 (Regroupement de tâches successives)**.** *Soit O un ordonnancement et $op_1, \ldots, op_a$ a tâches d'un même utilisateur exécutées successivement sur une même machine. L'action de remplacer $op_1, \ldots, op_a$ par une seule tâche notée $op_{1\rightarrow\ldots\rightarrow a}$ de même durée que la somme des durées des tâches $op_1, \ldots, op_a$ s'appelle le regroupement des tâches $op_1, \ldots, op_a$.*

Définition 4.17. *Un critère est dit* indépendant au regroupement de tâches *si, lorsque sont regroupées k tâches successives d'un même utilisateur en une seule tâche, la valeur du critère ne change pas.*

On recherche un critère de comparaison entre utilisateurs indépendant au regroupement de tâches successives ou dit autrement ces ordonnancements doivent être équivalents ; pour un site ce qui compte ce sont les intervalles de temps accordés aux tâches et non le découpage interne de ces intervalles de temps par l'utilisateur. Ainsi un site fournit des créneaux horaires dans lesquels les utilisateurs éxécutent leurs tâches. Un utilisateur ne changera pas la valeur de son critère s'il remplace ces tâches par une seule exécutée au même moment ou inversement s'il découpe une tâche en plusieurs exécutées à la suite. De même, deux utilisateurs qui utilisent des machines identiques pendant la même période de temps sont considérés comme ayant reçu la même qualité de service, indépendamment du découpage de leurs tâches sur ces machines.

Par exemple, la somme des temps de séjour n'est pas insensible au regroupement. Intuitivement, on comprend qu'ajouter les temps d'attente des tâches lorsque c'est le même utilisateur qui attend

n'a pas beaucoup de sens.

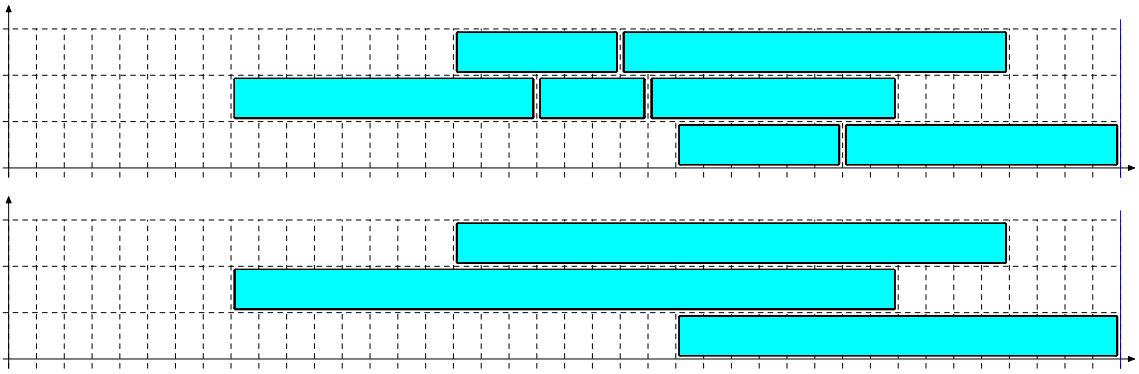


Fig. 4.6.1 Regroupement de tâches

Cela entraîne donc que le critère dépend seulement de qui occupe quelles machines à un instant donné et non de quelle tâche s'exécute sur une machine à un instant donné.

Une autre propriété souhaitable est que si on insère un temps d'attente entre 2 tâches successives du même utilisateur on s'attend à ce que la métrique diminue. Proposons quelques définitions.

Définition 4.18 (Tâche active, Présence d'une tâche)**.** *Une tâche i est* active *à l'instant $t \in \mathbb{R}$ si $t_i \leq t \leq t_i + p_i$. Elle est* présente *dans l'intervalle $[r_i, t_i + p_i]$.*

Un utilisateur est présent *à l'instant t s'il a au moins une tâche présente à t.*

La présence d'un utilisateur est l'union des intervalles de présence de ses tâches. On note $LIP_U(O)$ la liste des intervalles de présence de l'utilisateur U dans l'ordonnancement O.

Définition 4.19 (Temps de présence, Intervalle de présence)**.** *Le temps de présence d'un utilisateur U_h est la somme des longueurs des intervalles de présence de l'utilisateur, il est noté TP_h.*

Exemple

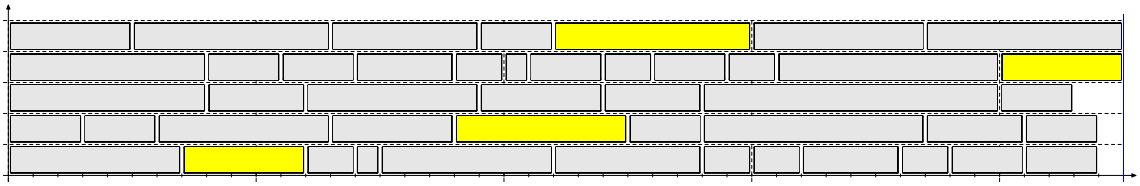


Fig. 4.6.2 Ordonnancement avec 5 machines.

Considérons l'ordonnancement de la figure 4.6.2, avec les tâches suivantes pour l'utilisateur jaune :

$r_1 = 0$	$p_1 = 5$	$t_1 = 7$
$r_2 = 16$	$p_2 = 7$	$t_2 = 18$
$r_3 = 20$	$p_3 = 8$	$t_3 = 22$
$r_4 = 39$	$p_4 = 5$	$t_4 = 40$

Les intervalles de présence des tâches sont alors :
$[0, 12]$
$[16, 25]$
$[20, 30]$
$[39, 45]$
La liste d'intervalle de présence de l'utilisateur est alors $([0, 12], [16, 30], [39, 45])$ avec un temps de présence de 32.

Propriétés des intervalles de présence :

- $LIP_U(O_1 \cup O_2) = LIP_U(O_1) \cup LIP_U(O_2)$
- $LIP_U(O_1) \subset LIP_U(O_1 \cup O_2)$
- Ainsi si $LIP_U(O_1) \subset LIP_U(O_2)$, on a pour tout O', $LIP_U(O_1 \cup O') \subset LIP_U(O_2 \cup O')$.

Définition 4.20 (Sessions)**.** *L'utilisateur U est présent dans une liste d'intervalle de présence $LIP_U(O)$ pour l'ordonnancement O. Un intervalle maximal de présence de $LIP_U(O)$ est appelé une* session*.*

Ainsi une session pour un utilisateur est un intervalle de temps maximal dans lequel l'utilisateur a au moins une tâche en attente ou en exécution.

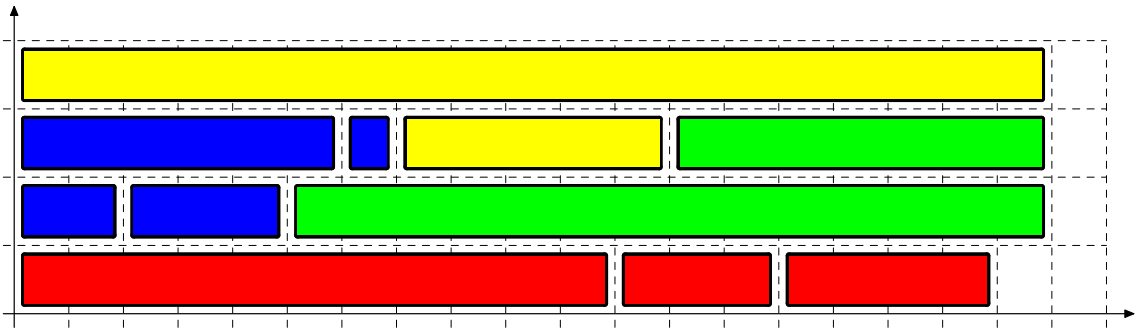


Fig. 4.6.3 Ordonnancement avec 4 utilisateurs et 4 machines.

La figure 4.6.3 donne un exemple d'ordonnancement avec 4 utilisateurs et 4 machines. Les tâches arrivent toutes en début de période. Les durées respectives des tâches sont :
- pour l'utilisateur Bleu : $\{1, 2, 3, 6\}$
- pour l'utilisateur Jaune : $\{19, 5\}$
- pour l'utilisateur Rouge : $\{11, 3, 4\}$
- pour l'utilisateur Vert : $\{7, 14\}$

Les intervalles de présence des utilisateurs sont :
- pour l'utilisateur Bleu : $[0, 7]$
- pour l'utilisateur Jaune : $[0, 19]$
- pour l'utilisateur Rouge : $[0, 18]$
- pour l'utilisateur Vert : $[0, 19]$

Définition 4.21 (Débit d'entrée, Débit d'exécution, Débit ponctuel)**.** *Rappelons que le débit d'entrée est égal au rapport entre la quantité de calcul des tâches reçues pendant une pé-*

riode et cette période, il ne dépend pas de l'ordonnancement effectué mais seulement des caractéristiques des tâches.

Le débit d'exécution est égal au rapport entre la quantité de calcul et le temps de présence *de l'utilisateur.*

Le débit ponctuel accordé à un utilisateur est la somme des vitesses des machines qui exécutent une tâche de cet utilisateur à un instant t.

Ainsi le débit d'entrée est aussi le nombre moyen de machines utilisées sur toute une période.

Le débit d'exécution est toujours supérieur au débit d'entrée. En effet, comme le temps de présence est toujours inférieur ou égal à la période et que le débit d'entrée est égal au rapport entre la quantité de calcul soumise par période et la période, le débit d'exécution est alors supérieur ou égal au débit d'entrée.

Lemme 4.12. *Il existe un instant t tel que le nombre de machines utilisées par l'utilisateur est supérieur ou égal à son débit d'exécution. Si un utilisateur utilise M_h machines, son débit d'exécution sera donc inférieur ou égal à M_h.*

Démonstration. En effet, divisons l'intervalle en intervalles de longueur ϵ. Soit $[k\epsilon, (k+1)\epsilon]$ le k^e intervalle, notons Q_k la quantité de calcul exécutée dans cet intervalle par U_h et $\pi_k = 1$ si l'utilisateur est présent dans cet intervalle ou $\pi_k = 0$ sinon. Alors on a :

$$\text{Debit execution}_h = \frac{Q_1 + Q_2 + \dots}{(\pi_1 + \pi_2 + \dots)\epsilon} = bar_{i|\pi_i=1}(\{\frac{Q_i}{\epsilon}, \epsilon\})$$

Donc si le débit d'exécution de l'utilisateur h est d'au moins M c'est qu'il existe un i^e intervalle tel que $\frac{Q_i}{\epsilon} \geq M$. Comme ϵ peut être pris aussi petit que souhaité il existe un instant t tel qu'au moins M machines sont utilisées par U_h. ∎

Ainsi, le débit d'exécution est la moyenne du nombre de machines utilisées sur les intervalles où l'utilisateur est présent. Il est équivalent au nombre de machines dans le cas de machines identiques ou à la somme des vitesses des machines allouées à l'utilisateur dans le cas où les machines sont hétérogènes.

Par conséquent, un utilisateur qui a à sa disposition plusieurs systèmes pour traiter ses données pourra les comparer selon le débit offert. Par exemple si un utilisateur qui doit traiter des calculs de courte durée, il ne sera pas forcément interessé par des gros systèmes avec plusieurs utilisateurs où les calculs sont planifiés sur des mois et pourrait préférer un système de petite taille avec peu d'utilisateurs mais avec aussi peu d'attente.

Notons qu'il peut y avoir une grande marge entre le débit d'entrée et le débit d'exécution, par exemple un utilisateur peut avoir M tâches de durée 1 et toutes les envoyer périodiquement à une date fixe. Dans ce cas s'il est possible d'exécuter toutes ses tâches à cette date son débit d'exécution sera de M alors que sont débit d'entrée est de $\frac{M}{T}$.

Considérons un ordonnancement où les opérations des utilisateurs U_1 et U_2 sont rassemblées : $U = U_1 \cup U_2$, alors $IP_U = IP_{U_1} \cup IP_{U_2}$ donc

$$debit_U = \frac{Q_U}{\|IP_U\|} = \frac{Q_{U_1} + Q_{U_2}}{\|IP_{U_1}\| + \|IP_{U_2}\| - \|IP_{U_1} \cap IP_{U_2}\|} \geq \frac{Q_{U_1} + Q_{U_2}}{\|IP_{U_1}\| + \|IP_{U_2}\|}$$

et

$$bar((debit_{U_1}, \|IP_{U_1}\|), (debit_{U_2}, \|IP_{U_2}\|)) \leq debit_U$$

On en conclut en particulier que le débit d'un groupe est supérieur ou égal au débit minimum des membres de ce groupe :

$$debit_U = bar((debit_{U_1}, \|IP_{U_1}\|), (debit_{U_2}, \|IP_{U_2}\|)) \frac{\|IP_{U_1}\| + \|IP_{U_2}\|}{\|IP_{U_1} \cup IP_{U_2}\|}$$

Soit pour un groupe :

$$debit_{\bigcup_{U_i}} = bar(\{(debit_{U_i}, \|IP_{U_i}\|)\}) \frac{\sum_i \|IP_{U_i}\|}{\|\bigcup_{U_i} IP_{U_i}\|}$$

Or, pour N utilisateurs du groupe on a $1 \leq \frac{\sum_i \|IP_{U_i}\|}{\|\bigcup_{U_i} IP_{U_i}\|} \leq N$, ainsi :

$$bar(\{(debit_{U_i}, \|IP_{U_i}\|)\}) \leq debit_{\bigcup_{U_i}} \leq N bar(\{(debit_{U_i}, \|IP_{U_i}\|)\})$$

Que peut-on dire si on alloue exclusivement à chaque utilisateur autant de machines que son débit d'entrée (partie supérieure) ?

Tout d'abord ceci n'est pas toujours possible car cela peut demander plus de machines au total que de machines disponibles. Mais lorsque cela l'est quelles sont les conséquences sur les débits obtenus pour chaque utilisateur ? D'après les lemmes précédents le débit d'exécution de chaque utilisateur sera compris entre son débit d'entrée et la partie supérieure de son débit d'entrée qui est le nombre de machines accordées.

La somme des débits d'entrée doit être inférieure au nombre total de machines, on a donc :

$$\begin{array}{rcl} \sum_{i \in U_h} p_i/\alpha_i & = & Debit_h^{entree} \\ \lceil Debit_h^{entree} \rceil & \leq & M_h \\ \sum_{U_h} \sum_{i \in U_h} p_i/\alpha_i & \leq & M \\ \sum_{U_h} \lceil Debit_h^{entree} \rceil & \leq & M + N \end{array}$$

$$M - N \leq \sum_{U_h} \lfloor Debit_h^{entree} \rfloor \leq M$$

On en conclut qu'il y a au moins $(\sum_{U_h} \lceil Debit_h^{entree} \rceil) - M$ machines qui sont partagées par plusieurs utilisateurs ; i.e. utilisées par au moins 2 utilisateurs au cours du temps. Ainsi, on déduit que :

Lemme 4.13. ***le nombre de machines partagées entre utilisateurs est inférieur à*** N.

Notons que si $\sum_{U_h} \lceil Debit_h^{entree} \rceil \leq M$ alors, il est possible de dédier $\lceil Debit_h^{entree} \rceil$ machines pour chaque utilisateur U_h. Supposons maintenant que tous les débits d'entrée soient entiers et de somme inférieure à M ; existe-t-il un ordonnancement qui dédie à chaque utilisateur son débit d'entrée ? Oui, car les tâches de chaque utilisateur sont ordonnançables avec un nombre de machines au moins égal à son débit d'entrée. De plus, comme la somme est inférieure à M le nombre de machines utilisées est inférieur à M.

4.6.2 Intervalles

Donnons quelques définitions et résultats utiles relatifs au critère.

Définition 4.22 (Ordres partiels sur les intervalles)**.**

$I_1 = [x_1, y_1], I_2 = [x_2, y_2]$

Ordre naturel induit :

$$I_1 \prec I_2 \Leftrightarrow y_1 < x_2$$

Remarque : $I_1 \prec I_2 \Rightarrow I_1 \cap I_2 = \varnothing$

Ordre d'inclusion :

$$I_1 \subseteq I_2$$

Cet ordre peut se généraliser aux listes d'intervalles.

4.6.3 Dominances

Nous allons présenter des résultats de dominance adaptés à la périodicité des tâches, dans le cas où le critère est "régulier".

Définition 4.23 (Ordre naturel sur les listes d'intervalles). $L_1 \prec L_2 \Leftrightarrow \forall I \in L_1, \exists I' \in L_2, I \subset I'$

Définition 4.24 (Critère croissant pour l'ordre d'inclusion sur les listes d'intervalles). *Un critère f est croissant sur l'ordre d'inclusion sur les listes d'intervalles si et seulement si on a :* $L_1 \prec L_2 \Rightarrow f(L_1) \leq f(L_2)$

Les critères basés sur la somme (= temps de présence) ou la longueur maximale des longueurs des intervalles de présence sont croissants. En revanche, la somme des dates de fin des tâches n'est pas un critère croissant pour l'ordre d'inclusion sur les listes d'intervalles.

Présentons quelques critères croissants pour l'ordre d'inclusion sur les listes d'intervalles :

Temps de présence maximal : ou sa version multi-utilisateurs qui consiste à maximiser $\prec_{Lex}$ sur le vecteur $[-T_h]$.

Temps de présence total : c'est la somme des temps de présence des utilisateurs. On retrouve la somme des temps de séjour lorsqu'il y a seulement une tâche par utilisateurs.

Débit minimal : c'est un critère de maximisation. Le débit est le rapport entre la quantité de calcul et le temps de présence pour chaque utilisateur. La version multi-utilisateurs consiste à maximiser $\prec_{Lex}$ sur le vecteur $[\frac{Q_h}{T_h}]$.

Somme des débits : on applique à la quantité de calcul pour chaque utilisateur une pondération inversement proportionnelle au temps de présence. On retrouve le critère du *ralentissement* (ou stretch [Bender et al., 1998; Legrand et al., 2008]) lorsque chaque utilisateur n'a qu'une seule tâche.

Définition 4.25 (Dominance sur les ordonnancements). *Un ordonnancement O_1 est dominé par un ordonnancement O_2 si et seulement si quelque soit O', on a pour tout utilisateur U, $LIP_U(O_2 \cup O') \subset LIP_U(O_1 \cup O')$ (rappelons que LIP_U signifie Liste d'Intervalles de Présence de l'utilisateur U). En particulier on doit avoir $LIP(O_2) \subset LIP(O_1)$.*

Si O_1 est dominé par O_2 alors, il est aussi dominé pour tout critère croissant pour l'ordre d'inclusion sur les listes d'intervalles de présence. Cela justifie l'utilisation de l'inclusion comme un ordre sur les listes d'intervalles, et donc sur les ordonnancements.

Notons que la borne inférieure des intervalles de présence est forcément égale à une date de disponibilité d'une des tâche de l'utilisateur. Ces bornes inférieures sont donc fixes pour les intervalles de présence des tâches. Cela justifie l'utilisation de l'inclusion comme un ordre sur les listes d'intervalles, et donc sur les ordonnancements.

Nous proposons d'utiliser les critères basés sur les intervalles de présence pour comparer un ordonnancement entre plusieurs utilisateurs car, par construction, ce critère est invariant par regroupement de tâches.

Remarquons que le critère des intervalles de présence devient celui du temps de séjour pour le cas où chaque utilisateur a une seule tâche. De plus, le temps de présence pondéré par la quantité de calcul traitée est équivalent à un nombre de machines.

4.7 Calcul des intervalles de présence minimale

Dans cette partie nous donnerons des méthodes pour calculer les intervalles de présence minimale des utilisateurs. Ces intervalles de présence minimale donnent une borne du temps de présence de chaque utilisateur. Cette borne permet de considérer un nombre de solutions plus restreint lors de la résolution.

L'évaluation d'un ordonnancement périodique s'effectue sur la partie périodique et non sur la partie transitoire. Les intervalles de présence sont donc calculés à partir du motif et sont répétés périodiquement.

Définition 4.26 (Paquets d'un utilisateur)**.** *La tâche i et la tâche j d'un utilisateur U_h appartiennent à un même paquet dans un ordonnancement donné si elles se trouvent dans un même intervalle de présence pour cet utilisateur dans cet ordonnancement.*

Un pâquet de tâche d'un utilisateur pour un ordonnancement donné est un ensemble de tâches qui appartiennent à un intervalle de présence pour cet ordonnancement.

Il existe des tâches qui ne peuvent jamais être séparées : soient deux tâches i et j telles que $r_i + p_i \geq r_j$ alors ces tâches ne sont pas séparables ; elles seront toujours dans le même intervalle de présence et sont donc dans le même paquet de tâches quelle que soit la façon de les placer dans un ordonnancement quelconque.

Cette remarque amène à la notion de paquets élémentaires :

Définition 4.27 (Tâches liées, Paquets élémentaires)**.** *Deux tâches sont* liées *sur M machines si quel que soit l'ordonnancement ces deux tâches sur M machines sont toujours dans le même paquet : elles sont indissociables quelle que soit la façon de placer ces tâches, elles sont toujours dans le même intervalle de présence.*

Un ensemble maximal de tâches liées constitue un paquet élementaire*.*

Les paquets de tâches d'un ordonnancement donné sont composés de paquets élémentaires de tâches.

Lemme 4.14 (Décomposition en paquets élémentaires)**.** *Dans un ordonnancement donné, un paquet de tâches se décompose en paquets élémentaires.*

Démonstration. Effectivement, soit un ordonnancement donné, si un paquet contient une tâche appartenant à un paquet élémentaire alors par définition il contient aussi toutes les tâches de ce paquet élémentaire. Ainsi, tout paquet de tâches d'un ordonnancement particulier est l'union disjoint de paquets élémentaires. ∎

Définition 4.28 (Utilisateurs permanents). *Un utilisateur est dit permanent s'il est présent pendant toute la période quelle que soit la façon de placer ses tâches.*

On peut déduire si certains utilisateurs sont permanents en calculant les paquets élémentaires (c'est suffisant mais non nécessaire). Si un utilisateur est permanent alors, on peut ne pas inclure son intervalle de présence (qui est alors constant) dans la fonction objectif globale.

Lemme 4.15. *Les paquets élémentaires pour $M' > M$ machines sont inclus dans ceux pour M machines.*

Démonstration. En effet un ordonnancement sur M est aussi un ordonnancement sur $M' > M$ machines. Par conséquent si deux tâches sont liées sur M' machines elles sont liées sur M machines aussi d'où le résultat. ■

4.7.1 Bornes sur le critère : intervalles minimaux de présence

4.7.1.a Borne "$M = \infty$"

Par conséquent, au regard du lemme précédent, les paquets élémentaires pour M machines sont composés des paquets élémentaires obtenus pour un nombre infini de machines. Or pour un nombre infini de machines, il est possible de caler chaque tâche sur sa date de disponibilité. Dans ce cas, les paquets obtenus sont minimaux. Il suffit alors de calculer ces paquets pour obtenir les paquets élémentaires pour un nombre infini de machines.

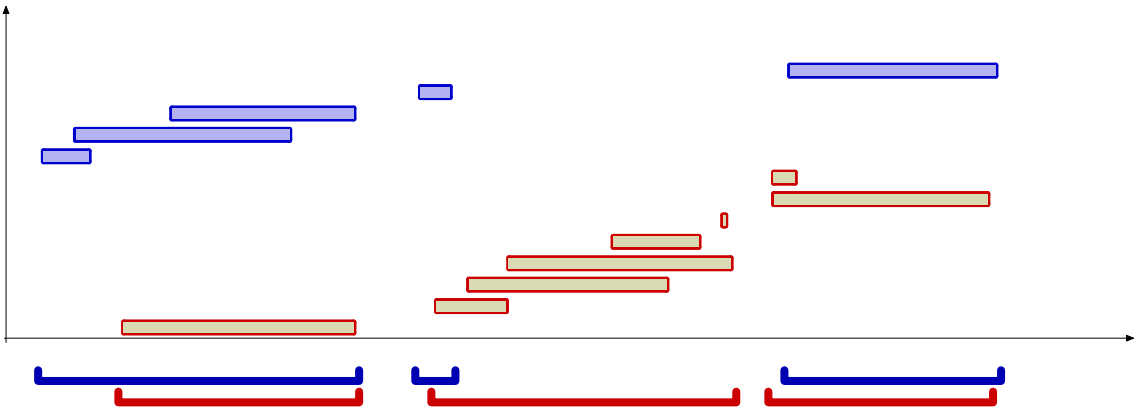


Fig. 4.7.1 $m = \infty$, 2 utilisateurs et intervalles de présence ci-dessous.

Voici un algorithme de calcul des paquets élémentaires :

1. Trier les tâches par date de disponibilité
2. Pour chaque tâche op
   (a) Si $[\text{op}.dispo, \text{op}.dispo + \text{op}.duree] \cap [paquet.min, paquet.max] \neq \varnothing$ alors insérer op dans le paquet courant
   (b) sinon ajouter le paquet courant à la liste des paquets en construction et le nouveau paquet courant contient op.
3. Retourner la liste des paquets

Où $paquet.max = \max_{\text{op} \in paquet}(\text{op}.dispo + \text{op}.duree)$ et $paquet.min = \min_{\text{op}} \text{op}.dispo$.

4.7.2 Borne "$M = 1$"

L'idée est de remplacer chaque tâche par une tâche de durée divisée par M et de caler les tâches à gauche sur une seule machine. Ceci est basé sur le fait qu'on ne peut écouler la quantité de calcul Q en une durée de moins de Q/M.

Théorème 4.16. *Les listes d'intervalles de présence obtenues sont alors forcément incluses dans les listes d'intervalles de présence de tout ordonnancement de ces tâches.*

Démonstration. Notons $LIP_U^{m=1}$ la liste d'intervalles obtenue par la borne $m = 1$.

Supposons que ce ne soit pas une borne, dans ce cas il existe un ordonnancement O avec un utilisateur U tel que $LIP_U^{m=1} \notin LIP_U(O)$. Il existe alors un intervalle $I = [a, b]$ de $LIP_U^{m=1}$ qui n'est inclus dans aucun intervalle de $LIP_U(O)$. I ne peut pas être $[0, T]$ car $[0, T] \subset LIP_U(O)$.

I est l'union des intervalles de présence d'un sous-ensemble E de tâches et a est forcément égal à une date de disponibilité d'une tâches op. Soit $I' = [a', b']$ l'intervalle de présence de O qui contient ce sous-ensemble de tâches. On a $a' \leq a$, puisque op appartient à E.

Comme $I \nsubseteq I'$ c'est que $b > b'$. Or cela est impossible puisque l'union des intervalles de présence des tâches de E est inclus dans I' et qu'il est impossible d'écouler la quantité de calcul Q^E de E en moins de $Q^E/m = b - a$. ∎

Remarquons encore que dans le cas périodique le fait de caler les tâches à gauche n'est pas immédiat puisque les tâches d'une période précédente peuvent décaler les tâches de la période en cours.

Ainsi les deux approches $M = 1$ et $M = \infty$ donnent chacune des listes d'intervalles de présence minimale par utilisateur dont l'union impose une liste d'intervalles par utilisateur où chacun est certain d'être présent.

4.7.3 Présentation du critère en lien avec l'équité

On cherche à être le plus équitable possible entre la quantité de calcul et le temps de présence par utilisateur. Cela se traduit par le fait de maximiser l'ordre Leximin sur le vecteur $[Q_h/T_h]$. Notons que s'il existe une constante ω telle que $\forall h, T_h = \omega Q_h$ avec le vecteur $[T_h]$ non dominé alors cette solution serait la plus équitable.

4.7.4 Remarques

Si chaque tâche appartient à un utilisateur différent, le problème de la somme des temps de présence est équivalent au problème de minimiser la somme des C_i avec dates de disponibilités.

S'il n'y a qu'un utilisateur, le problème revient à minimiser successivement le makespan avec des dates de disponibilités. Le problème mono-utilisateur est composé d'une suite de problèmes de Makespan ; car ce problème revient à minimiser son temps de présence qui est la somme des Makespan des différents paquets. En effet à partir du moment où deux tâches sont séparables il faut minimiser le Makespan du premier ensemble de tâches puis minimiser le Makespan du suivant et ainsi de suite. L'algorithme d'approximation de Shmoys [Shmoys et al., 1995] permet de construire un ordonnancement avec un coefficient d'approximation 2 fois moins bon à partir d'un algorithme

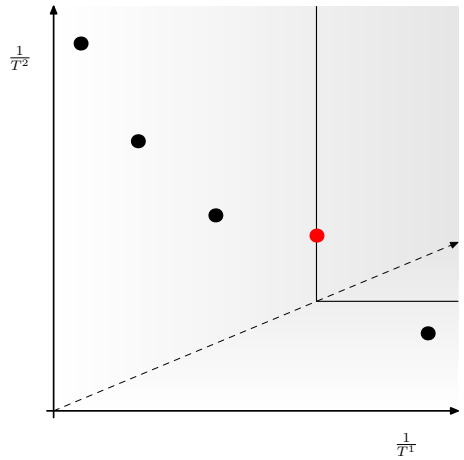


Fig. 4.7.2 Les axes représentent l'inverse des temps de présence de 2 utilisateurs. La quantité de calcul Q_h de chacun des utilisateurs est connue. La droite en pointillée correspond à Q_h/T_h constant. Les lignes de niveaux du Leximin pondéré sont représentées par un dégradé. La solution la plus équitable pour $[Q_h/T_h]$ est le point rouge.

approché pour le problème sans dates de disponibilités pour le Makespan. La technique du "Branch and Bound" donne de bons résultats pour ce problème.

Avec plusieurs utilisateurs et en supposant connaître les paquets élémentaires, le problème revient à savoir comment fusionner ces paquets au mieux selon le critère considéré.

4.8 Représentation en graphe d'intervalles

Selon le nombre de machines disponibles, le problème peut devenir infaisable ou au contraire trivial. Nous allons rechercher des conditions sur le nombre de machines nous permettant de déterminer si un problème est impossible ou simple à résoudre.

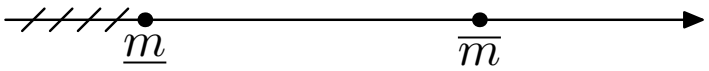


Fig. 4.8.1 Nombre de machines

4.8.1 Calcul du nombre de machines critique

Le nombre de machines critique est le nombre de machines minimum pour que le problème soit réalisable. Notons le $\underline{m}$. D'après la condition de réalisabilité on a $\underline{m} = \lceil \sum_{U_i,j} p_{i,j}/\alpha_{i,j} \rceil$.

Il est possible de se restreindre à chaque utilisateurs : si on considère le sous-ordonnancement obtenu en ne conservant que les tâches d'un seul utilisateur U_i, cet ordonnancement est faisable et utilise au moins $\lceil m_{U_i} \rceil$ machines avec $\underline{m_{U_i}} = \sum_j p_{i,j}/\alpha_{i,j} \leq m_{U_i}$.

Si $\sum_{U_i} m_{U_i}$ est voisin de M, étant donné que chaque utilisateur doit utiliser au moins $\lceil m_{U_i} \rceil$ machines, la résolution s'effectue sur quelques machines partagées par plusieurs utilisateurs. Notons

aussi que le débit de U_i est forcément supérieur à m_{U_i} (puisque le temps de présence de l'utilisateur est inférieur à la durée de la période).

4.8.2 Calcul du nombre minimal de machines pour caler chaque tâche à sa date de disponibilité

Définition 4.29 ($\overline{m}$). *Notons $\overline{m}$ le nombre minimum de machines nécessaires pour placer toutes les tâches à leurs dates de disponibilité.*

Ainsi lorsque $M \geq \overline{m}$, il est possible de placer chacune des tâches sur sa date de disponibilité, il s'agit alors de l'ordonnancement optimal. Nous verrons comment calculer ce nombre dans la suite, mais tout d'abord voici quelques résultats nécessaires sur les listes d'intervalles.

4.8.2.a Liste d'intervalles

Soit L un ensemble d'intervalles de la forme $[a_i, b_i]$.

Définition 4.30 (Encombrement de L à un instant t). *noté $\phi(L,t)$, est le nombre d'intervalles de L qui contiennent t.*

Définition 4.31 (Volume de L). *noté $\Phi(L)$, est l'encombrement maximal de L au cours du temps, soit $\max_t \phi(L,t)$.*

Définition 4.32 (Recouvrement). *On dit que L ne se recouvre pas si chacun de ses intervalles est disjoint.*

Définition 4.33 (Partition de L). *Une partition de L est un ensemble de liste d'intervalles L_i tel que $L = \bigcup_i L_i$ et chaque L_i ne se recouvre pas.*

Théorème 4.17 (Partitionnement de L). *Pour partitionner L, il faut un nombre de sous-ensembles égal à son volume et ce nombre est suffisant.*

Démonstration. En effet, soit t l'instant où $\phi(L,t) = \Phi(L)$; puisque les intervalles qui contiennent t sont au nombre de $\phi(L,t)$, le nombre de sous-ensembles nécessaire est au moins égal au volume de L.

Afin de montrer que ce nombre est suffisant, considérons l'algorithme suivant :

Fonction CompareIntervalles

Data : ($[a_1, b_1]$, $[a_2, b_2]$)
begin
  return $a_1 \leq a_2$;

Algorithme 3: Partionnement

Data : Liste d'intervalles L
begin
  $L' \leftarrow$ Tri(L, CompareIntervalles) ;
  Création de $\Phi(L)$ ensemble L_i vides. ;
  foreach $[a,b] \in L'$ **do**
    Insère $[a,b]$ dans le premier L_i tel que $\phi(L_i, a) = 0$;

Remarquons qu'étant donné que les intervalles sont ajoutés dans l'ordre de leur arrivée, si $\phi(L_i, a) = 0$ alors $\forall t \geq a, \phi(L_i, t) = 0$ et l'intervalle $[a, b]$ peut être ajouté dans L_i.

Cet algorithme utilise au plus $\Phi(L)$ sous-ensembles : en effet supposons le contraire, alors puisque les intervalles sont insérés dans l'ordre de leur borne inférieure c'est qu'il existe un a tel que $\phi(L, t) > \Phi(L)$, ce qui n'est pas possible.

Ce nombre est donc aussi suffisant. ∎

Théorème 4.18. *$\overline{m}$ est le nombre maximum d'intervalles $[r_{i,j}, r_{i,j} + p_{i,j}]$ se recoupant un instant donné.*

Démonstration. Soit L l'ensemble des intervalles $[r_{i,j}, r_{i,j} + p_{i,j}]$, alors L peut se partitionner en $\Phi(L)$ sous-ensemble d'intervalles disjoints et ce nombre est maximal. Si on associe une machine par sous-ensemble on obtient un placement réalisable des tâches. Comme $\Phi(L)$ est le nombre maximum d'intervalles se recoupant à un instant donné, on obtient le résultat. ∎

Théorème 4.19. *Soit un ordonnancement donné, le nombre minimum de machines pour réaliser cet ordonnancement est le nombre maximum d'intervalles $[t_{i,j}, t_{i,j} + p_{i,j}]$ se recoupant un instant donné.*

Démonstration. La preuve est identique au cas précédent : ce nombre est minimal car il existe un instant où $\Phi(L)$ tâches s'exécutent en même temps et ce nombre est suffisant car il est possible de partitionner les tâches. ∎

On peut considérer le problème sous un autre angle : partons de l'ordonnancement qui exécute toutes les tâches à leur date de disponibilité. Cet ordonnancement utilise $\overline{m}$ machines. Ensuite, on retarde l'exécution des tâches afin d'occuper moins de M machines, avec pour objectif de minimiser les intervalles de présence. Cette approche a déjà été considérée dans des problèmes avec partie obligatoire, voir [Poder et al., 2004].

Il est aussi possible de définir un nombre de machines critique par utilisateur : si on réserve plus de $\overline{m}_{U_i}$ machines à un utilisateur on cale toutes ses tâches à leur date de disponibilité. Notons aussi que le débit de U_i est forcément inférieur à $\overline{m}_{U_i}$ (puisque le temps de présence de l'utilisateur est supérieur à son temps de présence dans le cas $m = \infty$.) Ces deux nombres donnent des bornes pour le débit de l'utilisateur. De façon heuristique, on peut savoir a qui allouer en priorité pour le leximin, sachant que la borne inférieure du temps de présence n'est pas forcément atteignable, mais que la borne supérieure l'est.

4.8.3 Remarques sur le nombre de machines nécessaires pour des tâches avec marge (ou date butoir)

Chaque tâche d'un utilisateur appartient à un intervalle I de la liste des intervalles de présence minimale. Si on déplace l'intervalle de cette tâche tout en restant dans I cela ne change pas la présence de cet utilisateur. On peut donc voir la fin de I comme une date butoir (ou une marge) pour la tâche ; si on veut obtenir une liste d'intervalles de présence donnée on obtient un problème de faisabilité avec m machines et dates butoirs par tâches. Ainsi, il existe un nombre de machines $\hat{m} \leq \overline{m}$ pour lequel on puisse obtenir pour chaque utilisateur sa liste d'intervalles de présence minimale.

A partir de la liste des intervalles de présence minimale, il faudra choisir certaines tâches qui seront calées à leur disponibilité et déplacer les autres tâches pour utiliser au plus m machines et minimiser les intervalles de présence. Choisir les tâches qui débutent un intervalle de présence dans la liste des intervalles de présence peut être un bon point de départ.

Si on veut atteindre une liste d'intervalle de présence donnée par utilisateur, nous avons alors des marges (ou dates butoirs) par tâche :

Définition 4.34 (Date butoir). *Une date butoir d_i est la contrainte suivante : $t_i + p_i \leq d_i$.*

Soit une tâche de disponibilité r et de durée p avec une marge μ. Si $\mu < p$, alors $[r, r+p] \cap [r+\mu, r+\mu+p] \neq \varnothing$. La tâche est donc forcément présente dans l'intervalle $[r+\mu, r+p]$. On peut calculer à partir de ces intervalles un nombre de machines minimal nécessaire pour ces tâches avec marges et ainsi savoir si le nombre de machines n'est pas suffisant.

Avec ces intervalles de présence minimale par utilisateur on peut calculer la marge de manœuvre de chaque tâche. Si une tâche (r,p) est dans l'intervalle de présence $[a,b]$ elle a une marge de $b-(r+p)$ dans laquelle elle peut se déplacer sur le graphe $m=\infty$ sans changer ses intervalles de présence. Il faut noter que décider du placement sur M machines est NP-complet car équivalent à M machines avec une date butoir pour chaque tâche [Lawler, 1982].

Définition 4.35 (Partie obligatoire [Lahrichi, 1982; Poder et al., 2004]). *La partie obligatoire d'une tâche est l'intervalle égal à l'intersection de son intervalle de présence pris sur l'ensemble des ordonnancements réalisables.*

Dans notre cas, si une tâche (r,p) a une marge μ alors, si $\mu \leq p$ on est certain que cette tâche sera dans l'intervalle $[r+\mu, r+p] = [r, r+p] \cap [r+\mu, r+\mu+p]$. En calculant le partitionnement de ces intervalles, on obtient une borne sur le nombre de machines à avoir pour des tâches avec une marge de manœuvre.

4.8.4 Prétraitement et classification des problèmes

4.8.4.a Utilisateurs permanents

Définition 4.36 (Utilisateur permanent, utilisateur intermittent). *Un utilisateur est dit* permanent *s'il est toujours présent dans la période quel que soit la façon de placer ses tâches.*

Un utilisateur non permanent est dit intermittent.

Si tous les utilisateurs sont permanents tous les ordonnancements faisables sont équivalents en terme de temps de présence. Nous supposerons donc qu'il existe au moins un utilisateur non permanent.

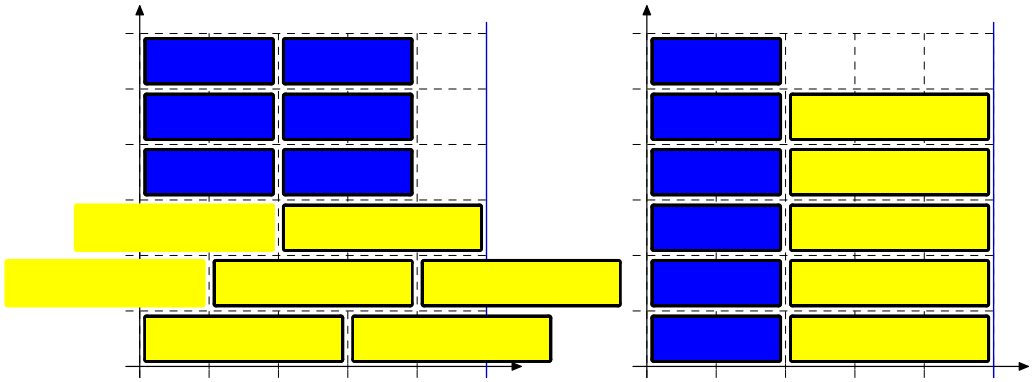
Classement des problèmes

On en déduit les propriétés suivantes :
- Si $m < \underline{m}$ le problème est infaisable.
- Si $m \geq \overline{m}$ il n'y a pas de conflits sur le nombre de machines, ainsi on peut caler chaque tâche sur sa date de disponibilité.
- Si après calcul des intervalles de présence tous les utilisateurs sont permanents on peut résoudre le problème avec un ordonnancements de type MacNaughton où on exécute les tâches de chaque utilisateur l'un après l'autre.

– Dans tout les autres cas la résolution n'est pas immédiate. Parmi ces problèmes certains semblent plus complexes que d'autres à résoudre.

Si un utilisateur est toujours présent quelque soit la façon de placer ses tâches (utilisateur permanent), il suffit de tenir le débit d'entrée qu'il demande en exécutant toutes ses tâches pour chaque période. Une idée peut être de dédier un nombre de machines suffisants aux utilisateurs permanents. Cependant les ordonnancements obtenus sont alors sous-optimaux, prenons l'exemple suivant :

Soient deux utilisateurs (bleu et jaune) et 6 machines, les tâches ont une période de 5. L'utilisateur bleu a 6 tâches de durée 2 et disponible à la date 0. L'utilisateur jaune a 5 tâches de durées 3 et en soumet une toutes les unités de temps. Ainsi, l'utilisateur jaune est un utilisateur permanent tandis que l'utilisateur bleu est un utilisateur non permanent. Considérons les ordonnancements des figures 4.8.2 ; le premier ordonnancement consiste à dédier 3 machines à l'utilisateur jaune, le vecteur des temps de présence est alors $[4, 5]$. Le second ordonnancement consiste à exécuter toutes les tâches de l'utilisateur bleu à la date 0 ; le vecteur des temps de présence est alors $[2, 6]$. On remarque que ce second ordonnancement domine le premier.



(a) 3 machines sont dédiées à l'utilisateur jaune.

(b) Les tâches de l'utilisateur bleu sont exécutées à la date 0.

Fig. 4.8.2 Dédier des machines aux utilisateurs permanents est sous-optimal.

Pour ce qui est du temps de présence, on peut ainsi diviser les utilisateurs en deux groupes :

– Les utilisateurs permanents : il suffit de placer toutes leurs tâches dans une période : l'important pour le site étant de pouvoir "tenir le rythme" de leur débit d'entrée, c'est le cas pour certains utilisateurs de production qui doivent traiter des données produites en permanence.
– Les utilisateurs intermittents ont un besoin de puissance de calcul en "pics" : ils ont une demande en ressources de calcul lors d'un intervalle de temps qu'ils espèrent le plus court possible.

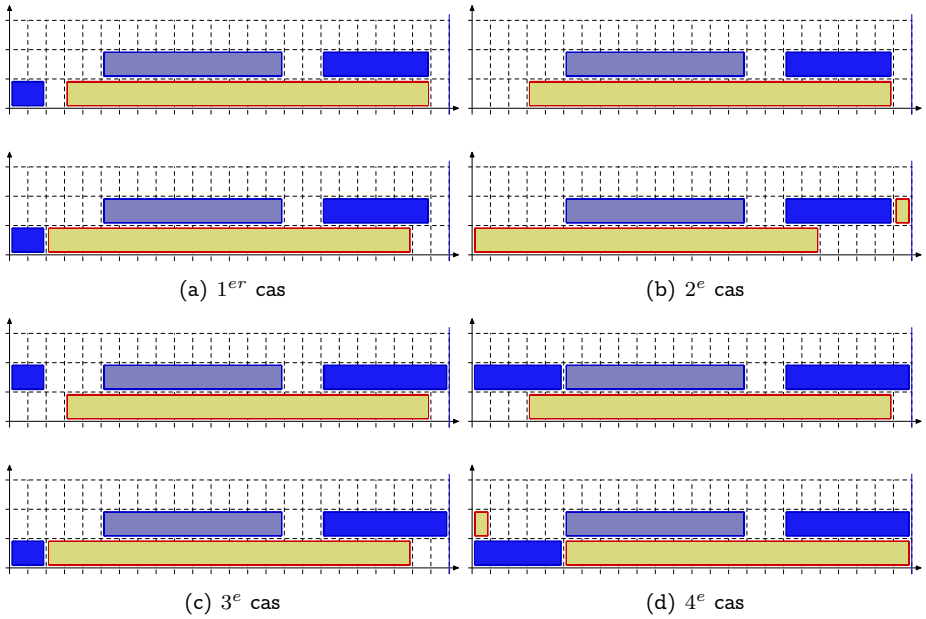
On constate que le même critère basé sur les temps de présence prend en compte ces deux catégories d'utilisation.

Il est possible d'ajouter une contrainte de seuil pour certains utilisateurs nécessitant simplement un débit acceptable, comme c'est le cas pour certains utilisateurs "interactifs" qui ont beaucoup de tâches de durée courtes, ils ont simplement besoin que les temps de réponse soient raisonnables.

Présentons un résultat de dominance pour les tâches des utilisateurs permanents :

Théorème 4.20 (Tâches des utilisateurs permanents). *Les ordonnancements pour lesquels les tâches des utilisateurs permanents sont calées sur la fin d'une tâche précédente sont dominants.*

Démonstration. Soit un motif d'ordonnancement périodique dans lequel une tâche O d'un utilisateur permanent U n'est pas calée sur la fin d'une autre tâche. Alors, il existe un temps d'oisiveté sur cette machine avant le début de la tâche O. Notons que nous parlons ici de machine au sens périodique, i.e. c'est le cycle sur lequel se situe la tâche O et peut ainsi regrouper plusieurs machines sur une période.



(a) 1^{er} cas (b) 2^{e} cas

(c) 3^{e} cas (d) 4^{e} cas

Fig. 4.8.3 Différents cas de figure

1^{er} *cas :* Supposons que sur cette machine il existe une autre tâche O' avant la tâche O, alors le fait d'avancer O en la calant sur O' ne change pas le temps de présence de l'utilisateur permanent U.

Supposons que sur cette machine il n'existe pas d'autres tâches avant la tâche O (i.e. puisque c'est périodique O est alors la seule tâche sur cette machine). Comme O n'est pas calée, il existe un temps d'oisiveté t sur cette machine.

2^{e} cas : S'il existe une machine M non vide et oisive au temps T, on peut recoller la machine de O à la fin de M, puis la fin de la machine de O au début de la machine M. Ainsi, on retrouve le 1^{er} cas.

3^{e} cas : Supposons que toutes les machines non vides ne soient pas oisives à T. Ainsi, il existe une tâche sur M qui traverse la période, soit t' sa date de fin. Si $t' - T \leq t$ alors, on peut déplacer le morceau de raccord sur la machine de O et on retrouve le 1^{er} cas.

4^{e} cas : Supposons que toutes les machines non vides ne soient pas oisives à T et que sur toutes les machines non vides on ait $t' - T > t$. Dans ce cas déplaçons le morceau de raccord sur la machine de

O, puis calons O et plaçons le morceau de raccord de O sur M, ce qui est rendu possible puisque la durée du morceau de raccord de O est de $t' - T - t$ et que $0 < t' - T - t < t' - T$. Ainsi, on retrouve le 1^{er} cas.

Par conséquent, caler les tâches des utilisateurs permanents sur la fin d'une autre tâche est dominant. ■

Ainsi lorsqu'on doit décider du placement d'une tâche après un temps d'oisiveté sur une machine, les tâches des utilisateurs intermittents sont prioritaires par rapport aux tâches des utilisateurs permanents.

4.9 Résolution exacte : formulation linéaire en nombres entiers

4.9.1 Modélisation avec phase transitoire

Nous proposerons deux modèles pour notre problème d'ordonnancement à exécution périodique. Dans le premier, nous modélisons toutes les tâches de la phase transitoire depuis la période de complète disponibilité jusqu'à l'apparition du motif. Dans le second, nous modélisons les tâches qui formeront un motif.

4.9.2 Modèle avec calcul de la phase transitoire.

Ce premier modèle peut aussi être utilisé en fixant un motif afin de rechercher une phase transitoire de durée minimale. Commençons par décrire l'ensemble des variables de ce modèle.

4.9.2.a Les variables du modèle

Notons $NbOpTotal = N^{op}\Gamma$ le nombre total d'opérations reçues par période. De même, notons $K_i^\Gamma = \Gamma T/\alpha_i$ le nombre d'occurrences de la tâche i pendant Γ périodes.

Variables de placement des tâches

Placement des tâches sur les machines : $m_{i,k,l} = 1$ si et seulement si la tâche $\langle i,k\rangle$ s'exécute sur la machine l et $m_{i,k,l} = 0$ sinon. Nombre de variables entières : $M \times NbOpTotal$.

Date de début de la tâche $\langle i,k\rangle$ *:* $t_{i,k}$ On peut définir une liste des dates de début d'exécution par tâches : $\langle t_{i,1}, t_{i,2}, t_{i,3}, \ldots\rangle$.

Variables auxiliaires du problème

Ces variables sont déduites des autres variables :

Disjonction entre machines : $y_{i_1,k_1,i_2,k_2,l} = 2$ si et seulement si la tâche $\langle i_1,k_1\rangle$ et la tâche $\langle i_2,k_2\rangle$ s'exécutent sur la même machine l. Sinon $y_{i_1,k_1,i_2,k_2,l} = 0$ ou 1. Cette variable se déduit de l'ensemble des variables $m_{i,k,l}$:

$$\forall l = 1\ldots M, \forall i_1 = 1\ldots N^{op}, i_2 = 1\ldots N^{op}, k_1 = 1\ldots K_{i_1}^\Gamma, k_2 = 1\ldots K_{i_2}^\Gamma, \langle i_1,k_1\rangle \neq \langle i_2,k_2\rangle$$

$$\left\{ \; y_{i_1,k_1,i_2,k_2,l} \quad = \quad m_{i_1,k_1,l} + m_{i_2,k_2,l} \right.$$

Tâches sur la même machine donnée $Z_{i_1,k_1,i_2,k_2,l} = 1$ si et seulement si la tâche $\langle i_1,k_1\rangle$ est exécutée sur la même machine l que $\langle i_2,k_2\rangle$ et $Z_{i_1,k_1,i_2,k_2,l} = 0$ sinon. Elle se déduit des autres variables :

$$\forall l = 1\ldots M, \forall i_1 = 1\ldots N^{op}, i_2 = 1\ldots N^{op}, k_1 = 1\ldots K_{i_1}^\Gamma, k_2 = 1\ldots K_{i_2}^\Gamma, \langle i_1,k_1\rangle \neq \langle i_2,k_2\rangle$$

$$\left\{ \begin{array}{rcl} Z_{i_1,k_1,i_2,k_2,l} & \geq & 0 \\ y_{i_1,k_1,i_2,k_2,l} - 1 & \leq & 2Z_{i_1,k_1,i_2,k_2,l} \\ 2Z_{i_1,k_1,i_2,k_2,l} & \leq & y_{i_1,k_1,i_2,k_2,l} \end{array} \right.$$

Cette contrainte doit assurer les choses suivantes :

– S'il existe une machine l telle que $y_{i_1,k_1,i_2,k_2,l} = 2$ alors $z_{i_1,k_1,i_2,k_2} = 1$
– sinon, si pour toutes les machines l $y_{i_1,k_1,i_2,k_2,l} = 0$ ou 1 alors $z_{i_1,k_1,i_2,k_2} = 0$

$$y_{i_1,k_1,i_2,k_2,l} - 1 \leq 2Z_{i_1,k_1,i_2,k_2,l} \quad (4.9.1)$$
$$2Z_{i_1,k_1,i_2,k_2,l} \leq y_{i_1,k_1,i_2,k_2,l} \quad (4.9.2)$$

Montrons l'équivalence. Le système est : $y - 1 \leq 2Z; 2Z \leq y$ et on veut démontrer que si $y = 2$ alors $Z = 1$ et si $y = 1$ ou $y = 0$ alors $Z = 0$.
– Si $Z = 0$, alors $y - 1 \leq 0; 0 \leq y$ donc $y = 0$ ou $y = 1$.
– Si $y = 1$, alors $0 \leq 2Z; 2Z \leq 1$ donc $Z = 0$.
– Si $y = 0$, alors $-1 \leq 2Z; 2Z \leq 0$ donc $Z = 0$.
– Si $Z = 1$, alors $y - 1 \leq 2; 2 \leq y$ et $y = 2$.
– Si $y = 2$, alors $1 \leq 2Z; 2Z \leq 2$ donc $Z = 1$.

Tâches sur la même machine $z_{i_1,k_1,i_2,k_2} = 1$ si et seulement si la tâche $\langle i_1, k_1\rangle$ est exécutée sur la même machine que $\langle i_2, k_2\rangle$ et $z_{i_1,k_1,i_2,k_2} = 0$ sinon. Elle se déduit des autres variables comme ceci :

$$\forall i_1 = 1 \dots N^{op}, i_2 = 1 \dots N^{op}, k_1 = 1 \dots K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1, k_1\rangle \neq \langle i_2, k_2\rangle$$

$$\left\{ z_{i_1,k_1,i_2,k_2} = \sum_{l=1}^{M} Z_{i_1,k_1,i_2,k_2,l} \right.$$

Précédences entre tâches : $x_{i_1,k_1,i_2,k_2} = 1$ si et seulement si la tâche $\langle i_1, k_1\rangle$ est effectuée avant la tâche $\langle i_2, k_2\rangle$. Comme a priori les périodes des tâches sont différentes les indices de périodes pour des tâches différentes n'ont aucun rapport entre eux et ne peuvent donc pas se comparer, du coup les variables x_{i,k_1,j,k_2} sont bien indexées par 4 indices.
On a (avec H une constante suffisamment grande) :

$$\forall i_1 = 1 \dots N^{op}, i_2 = 1 \dots N^{op}, k_1 = 1 \dots K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1, k_1\rangle \neq \langle i_2, k_2\rangle$$

$$\left\{ \begin{array}{rcl} x_{i_1,k_1,i_2,k_2} + x_{i_2,k_2,i_1,k_1} & = & 1 \\ t_{i_2,k_2} & \leq & t_{i_1,k_1} + x_{i_1,k_1,i_2,k_2} H \end{array} \right.$$

4.9.2.b *Contraintes*

Chaque tâche s'exécute sur une seule machine

Chaque tâche s'exécute sur une machine et une seule.

$$\forall i = 1 \dots N^{op}, \forall k = 1 \dots K_i^{\Gamma}, \forall l = 1 \dots M$$

$$m_{i,k,l} \in \{0, 1\}$$

$$\left\{ \sum_{l=1}^{M} m_{i,k,l} = 1 \right.$$

Dates de disponibilités

La tâche $\langle i,k\rangle$ doit commencer son exécution après sa date d'arrivée $r_{i,k}$ et avant la date de début d'exécution de la tâche $\langle i,k+1\rangle$.

$$\forall i = 1 \dots N^{op}, \forall k = 1 \dots K_i^{\Gamma}$$

$$\begin{cases} t_{i,k} & \geq & r_i + (k-1)\alpha_i \\ t_{i,k} & \leq & t_{i,k+1} \end{cases}$$

Chaque machine n'exécute qu'une tâche à un instant donné.

Lorsque deux tâches sont exécutées sur une même machine et que l'une est située avant l'autre, elles ne doivent pas se chevaucher. Soit H une constante suffisamment grande :

$$\forall l = 1 \dots M, i_1 = 1 \dots N^{op}, i_2 = 1 \dots N^{op}, k_1 = 1 \dots K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1,k_1\rangle \neq \langle i_2,k_2\rangle$$

$$\left\{ \; t_{i_1,k_1} + p_{i_1} \quad \leq \quad t_{i_2,k_2} + H(3 - x_{i_1,k_1,i_2,k_2} - m_{i_1,k_1,l} - m_{i_2,k_2,l}) \right.$$

Si les deux tâches $\langle i_1,k_1\rangle$ et $\langle i_2,k_2\rangle$ ne sont pas sur la même machine l alors :

- Si aucune tâches n'est sur la machine l alors $m_{i_1,k_1,l} + m_{i_2,k_2,l} = 0$
  La contrainte devient : $t_{i_1,k_1} + p_{i_1} \leq t_{i_2,k_2} + H(3 - x_{i_1,k_1,i_2,k_2})$ qui est trivialement verifiée quelle que soit la valeur de x_{i_1,k_1,i_2,k_2} à cause de H.
- Si une seule tâche est sur la machine l alors $m_{i_1,k_1,l} + m_{i_2,k_2,l} = 1$
  La contrainte devient : $t_{i_1,k_1} + p_{i_1} \leq t_{i_2,k_2} + H(2 - x_{i_1,k_1,i_2,k_2})$ qui est trivialement verifiée quelque soit la valeur de x_{i_1,k_1,i_2,k_2} à cause de H.
- Si chacune des tâches est sur la machine l alors $m_{i_1,k_1,l} + m_{i_2,k_2,l} = 2$ La contrainte devient : $t_{i_1,k_1} + p_{i_1} \leq t_{i_2,k_2} + H(1 - x_{i_1,k_1,i_2,k_2})$ qui est trivialement verifiée si $x_{i_1,k_1,i_2,k_2} = 0$, c'est à dire si $\langle i_1,k_1\rangle$ est après $\langle i_2,k_2\rangle$.

La contrainte devient : $t_{i_1,k_1} + p_{i_1} \leq t_{i_2,k_2}$
dans le cas où $x_{i_1,k_1,i_2,k_2} = 1$ c'est à dire si $\langle i_1,k_1\rangle$ est ordonnancé avant $\langle i_2,k_2\rangle$.

Exécution périodique des tâches :

La $K_i + 1^e$ occurrence de la tâche générique i s'exécute T unités de temps après le début d'exécution de la première occurrence de la tâche générique i. Soit dit autrement : entre le début de la tâche $\langle i,1+K_i\rangle$ et le début de $\langle i,1\rangle$, il y a T unités de temps :

$$\forall i = 1 \dots N^{op}, k = 1 \dots (\Gamma - 1) * K_i$$

$$\left\{ \; t_{i,k+K_i} = t_{i,k} + T \right.$$

Si deux tâches sont sur une même machine alors elles seront aussi les deux sur la même machine pour la période suivante.

$$\forall i_1 = 1 \dots N^{op}, i_2 = 1 \dots N^{op}, k_1 = 1 \dots K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1,k_1\rangle \neq \langle i_2,k_2\rangle$$

$$\left\{ \; z_{i_1,k_1,i_2,k_2} = z_{i_1,k_1+K_1,i_2,k_2+K_2} \right.$$

Contraintes de périodicité des précédences

Si deux tâches sont dans un certain ordre dans une période donnée, alors elles seront dans le même ordre dans les autres périodes :

$$\forall i_1 = 1 \dots N^{op}, k_1 = 1 \dots (\Gamma - 1) * K_{i_1}, k_2 = 1 \dots (\Gamma - 1) * K_{i_2}, i_2 = 1 \dots N^{op}$$

$$\{ \; x_{i_1,k_1,i_2,k_2} \quad = \quad x_{i_1,k_1+K_1,i_2,k_2+K_2}$$

Ces contraintes sont en fait induites par les contraintes précédentes.

Contraintes sur le motif

Ce groupe de contrainte assure que la dernière période contient effectivement toutes les tâches. Le modèle proposé jusqu'ici décrivait uniquement le processus d'arrivée périodique des tâches. On impose de chercher des solutions qui soient périodiques en exécution via des contraintes de motif. Il faut prendre en compte les tâches qui sont découpées entre périodes et qui doivent se retrouver au début de la période suivante, éventuellement sur des machines différentes.

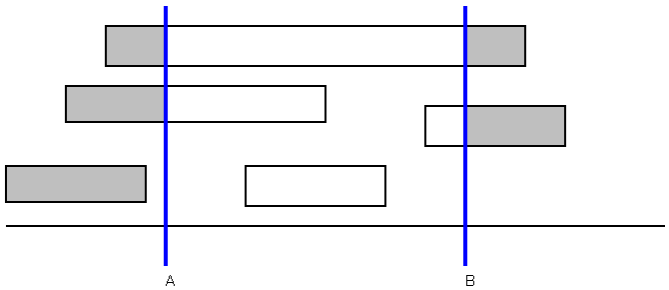


Fig. 4.9.1 $A = (\Gamma - 1)T, B = \Gamma T$

Notre idée est d'imposer la consommation des ressources pour chaque tâche dans la dernière période qui est l'intervalle $[(\Gamma - 1)T, \Gamma T]$.

Nous allons précalculer la consommation de chaque tâche. Notons $Qc_i = \frac{p_i T}{\alpha_i}$, Qc_i est donc une donnée du problème. Nous allons exprimer cette contrainte sur le motif pour la dernière période.

On veut calculer la somme de tous les morceaux de tâches appartenant à l'intervalle sachant que la date de début des tâches est $t_{i,k}$ et la fin est $t_{i,k} + p_i$.

La fonction $x \to \max(A, \min(B, x))$ permet de se limiter à la "fenêtre" $[A, B]$ comme le montre la figure 4.9.1. En effet lorsque $x > b$, alors $\max(A, B) = B$, si $x < a$ on obtient $\max(A, x) = A$, et si $A \leq x \leq B$ alors $\max(A, x) = x$.

Nous appliquons cette fonction à chaque tâche générique i et ses tâches $\langle i, k \rangle$ respectives d'intervalle d'exécution $[t_{i,k}, t_{i,k} + p_i]$:

$$Qr_i = \sum_k \max(A, \min(B, t_{i,k} + p_i)) - \max(A, \min(B, t_{i,k}))$$

Ce calcul s'effectue en linéarisant la fonction : $x \to \max(A, \min(B, x))$. Voyons maintenant comment linéariser cette fonction.

Calcul d'un minimum par programmation linéaire

On utilise la variable binaire η telle que $\eta = 1$ si $B \leq x$ et 0 sinon. Le programme linéaire suivant calcule alors le maximum entre A et x :

$$\begin{cases} y & = & \min(B,x) \\ \eta & \in & \{0,1\} \\ \eta H + B & \geq & x \\ (1-\eta)H + x & \geq & B \\ (1-\eta)H + y & \geq & B \\ y & \leq & B \\ \eta H + y & \geq & x \\ y & \leq & x \end{cases}$$

En effet, si $B \leq x$ alors $\eta = 1$ car $\eta H + B \geq x$ et on a alors $y = B = \min(B,x)$ car $y \leq B$ et $(1-\eta)H + y \geq B$. De même, si $x \leq B$ alors $\eta = 0$ car $(1-\eta)H + x \geq B$ ainsi $y = x = \min(B,x)$ car $y \leq x$ et $\eta H + y \geq x$.

Calcul d'un maximum par programmation linéaire

On utilise la variable binaire θ telle que $\theta = 1$ si $x \geq A$ et 0 sinon.

$$\begin{cases} y & = & \max(A,x) \\ \theta & \in & \{0,1\} \\ \theta H + A & \geq & x \\ (1-\theta)H + x & \geq & A \\ (1-\theta)H + x & \geq & y \\ y & \geq & A \\ \theta H + A & \geq & y \\ y & \geq & x \end{cases}$$

Ce programme linéaire est déduit du précédent en inversant les signes des variables et les inégalités.

Calcul de $Qr_{i,k}$

Notons le k^e terme de la somme :

$$Qr_{i,k} = \max(A, \min(B, t_{i,k} + p_i)) - \max(A, \min(B, t_{i,k}))$$

Soit

$$\begin{cases} y_{i,k}^{\min} & = & \min(B, t_{i,k}) \\ y_{i,k}^{\max} & = & \max(A, y_{i,k}^{\min}) \\ z_{i,k}^{\min} & = & \min(B, t_{i,k} + p_i) \\ z_{i,k}^{\max} & = & \max(A, z_{i,k}^{\min}) \end{cases}$$

Etape 1. Calcul de $y_{i,k}^{\min}$

$$\left\{\begin{array}{rcl} y_{i,k}^{\min} & = & \min(B, t_{i,k}) \\ \eta_{i,k} & \in & \{0,1\} \\ \eta_{i,k}H + B & \geq & t_{i,k} \\ (1-\eta_{i,k})H + t_{i,k} & \geq & B \\ (1-\eta_{i,k})H + y_{i,k}^{\min} & \geq & B \\ y_{i,k}^{\min} & \leq & B \\ \eta_{i,k}H + y_{i,k}^{\min} & \geq & t_{i,k} \\ y_{i,k}^{\min} & \leq & t_{i,k} \end{array}\right.$$

Etape 2. Calcul de $y_{i,k}^{\max}$

$$\left\{\begin{array}{rcl} y_{i,k}^{\max} & = & \max(A, y_{i,k}^{\min}) \\ \theta_{i,k} & \in & \{0,1\} \\ \theta_{i,k}H + A & \geq & y_{i,k}^{\min} \\ (1-\theta_{i,k})H + y_{i,k}^{\min} & \geq & A \\ (1-\theta_{i,k})H + y_{i,k}^{\min} & \geq & y_{i,k}^{\max} \\ y_{i,k}^{\max} & \geq & A \\ \theta_{i,k}H + A & \geq & y_{i,k}^{\max} \\ y_{i,k}^{\max} & \geq & y_{i,k}^{\min} \end{array}\right.$$

Etape 3. Calcul de $z_{i,k}^{\min}$

$$\left\{\begin{array}{rcl} z_{i,k}^{\min} & = & \min(B, t_{i,k} + p_i) \\ \eta'_{i,k} & \in & \{0,1\} \\ \eta'_{i,k}H + B & \geq & t_{i,k} + p_i \\ (1-\eta'_{i,k})H + t_{i,k} + p_i & \geq & B \\ (1-\eta'_{i,k})H + z_{i,k}^{\min} & \geq & B \\ z_{i,k}^{\min} & \leq & B \\ \eta'_{i,k}H + z_{i,k}^{\min} & \geq & t_{i,k} + p_i \\ z_{i,k}^{\min} & \leq & t_{i,k} + p_i \end{array}\right.$$

Etape 4. Calcul de $z_{i,k}^{\max}$

$$\left\{\begin{array}{rcl} z_{i,k}^{\max} & = & \max(A, z_{i,k}^{\min}) \\ \theta'_{i,k} & \in & \{0,1\} \\ \hline \\ \theta'_{i,k}H + A & \geq & z_{i,k}^{\min} \\ (1-\theta'_{i,k})H + z_{i,k}^{\min} & \geq & A \\ (1-\theta'_{i,k})H + z_{i,k}^{\min} & \geq & z_{i,k}^{\max} \\ z_{i,k}^{\max} & \geq & A \\ \theta'_{i,k}H + A & \geq & z_{i,k}^{\max} \\ z_{i,k}^{\max} & \geq & z_{i,k}^{\min} \end{array}\right.$$

On impose alors d'avoir :

$$\left\{\begin{array}{rcl} Qr_{i,k} & = & z_{i,k}^{\max} - y_{i,k}^{\max} \\ Qr_{i,k} & = & \frac{p_i T}{\alpha_i} \end{array}\right.$$

4.9.2.c Formalisation

$$
\begin{cases}
\begin{cases}
\forall i = 1 \dots N^{op}, & k = 1 \dots K_i^{\Gamma}, \\
& \sum_{l=1}^{M} m_{i,k,l} = 1 \\
& t_{i,k} \geq r_i + (k-1)\alpha_i \\
& t_{i,k} \leq t_{i,k+1}
\end{cases} \\
\begin{cases}
\forall i = 1 \dots N^{op}, & k = 1 \dots (\Gamma - 1) * K_i, \\
& t_{i,k+K_i} = t_{i,k} + T
\end{cases} \\
\begin{cases}
\forall i_1 = 1 \dots N^{op}, & i_2 = 1 \dots N^{op}, k_1 = 1 \cdots = K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1, k_1 \rangle \neq \langle i_2, k_2 \rangle, \\
& x_{i_1,k_1,i_2,k_2} + x_{i_2,k_2,i_1,k_1} = 1 \\
& z_{i_1,k_1,i_2,k_2} = \sum_{l=1}^{M} Z_{i_1,k_1,i_2,k_2,l} \\
& t_{i_2,k_2} \leq t_{i_1,k_1} + x_{i_1,k_1,i_2,k_2} H
\end{cases} \\
\begin{cases}
\forall l = 1 \dots M, & i_1 = 1 \dots N^{op}, i_2 = 1 \dots N^{op}, k_1 = 1 \cdots = K_{i_1}^{\Gamma}, k_2 = 1 \dots K_{i_2}^{\Gamma}, \langle i_1, k_1 \rangle \neq \langle i_2, k_2 \rangle, \\
& t_{i_1,k_1} + p_{i_1} \leq t_{i_2,k_2} + H(3 - x_{i_1,k_1,i_2,k_2} - m_{i_1,k_1,l} - m_{i_2,k_2,l}) \\
& y_{i_1,k_1,i_2,k_2,l} = m_{i_1,k_1,l} + m_{i_2,k_2,l} \\
& Z_{i_1,k_1,i_2,k_2,l} \geq 0 \\
& y_{i_1,k_1,i_2,k_2,l} - 1 \leq 2Z_{i_1,k_1,i_2,k_2,l} \\
& 2Z_{i_1,k_1,i_2,k_2,l} \leq y_{i_1,k_1,i_2,k_2,l}
\end{cases} \\
\begin{cases}
\forall i_1 = 1 \dots N^{op}, & i_2 = 1 \dots N^{op}, k_1 = 1 \dots (\Gamma - 1) * K_{i_1}, k_2 = 1 \dots (\Gamma - 1) * K_{i_2}, \\
& x_{i_1,k_1,i_2,k_2} = x_{i_1,k_1+K_1,i_2,k_2+K_2} \\
& z_{i_1,k_1,i_2,k_2} = z_{i_1,k_1+K_1,i_2,k_2+K_2}
\end{cases} \\
\begin{cases}
\forall i = 1 \dots N^{op}, \\
\qquad p_i T / \alpha_i = \sum_{k=1}^{K_i^{\Gamma}} z_{i,k}^{\max} - y_{i,k}^{\max}
\end{cases} \\
\begin{cases}
\forall i = 1 \dots N^{op}, k = 1 \dots K_i^{\Gamma}, \\
A = (\Gamma - 1)T \\
B = \Gamma T \\
\begin{cases}
\eta_{i,k} \in \{0,1\} \\
\eta_{i,k} H + B \geq t_{i,k} \\
(1 - \eta_{i,k}) H + t_{i,k} \geq B \\
(1 - \eta_{i,k}) H + y_{i,k}^{\min} \geq B \\
y_{i,k}^{\min} \leq B \\
\eta_{i,k} H + y_{i,k}^{\min} \geq t_{i,k} \\
y_{i,k}^{\min} \leq t_{i,k}
\end{cases}
\quad
\begin{cases}
\theta_{i,k} \in \{0,1\} \\
\theta_{i,k} H + A \geq y_{i,k}^{\min} \\
(1 - \theta_{i,k}) H + y_{i,k}^{\min} \geq A \\
(1 - \theta_{i,k}) H + y_{i,k}^{\min} \geq y_{i,k}^{\max} \\
y_{i,k}^{\max} \geq A \\
\theta_{i,k} H + A \geq y_{i,k}^{\max} \\
y_{i,k}^{\max} \geq y_{i,k}^{\min}
\end{cases} \\
\begin{cases}
\eta'_{i,k} \in \{0,1\} \\
\eta'_{i,k} H + B \geq t_{i,k} + p_i \\
(1 - \eta'_{i,k}) H + t_{i,k} + p_i \geq B \\
(1 - \eta'_{i,k}) H + z_{i,k}^{\min} \geq B \\
z_{i,k}^{\min} \leq B \\
\eta'_{i,k} H + z_{i,k}^{\min} \geq t_{i,k} + p_i \\
z_{i,k}^{\min} \leq t_{i,k} + p_i
\end{cases}
\quad
\begin{cases}
\theta'_{i,k} \in \{0,1\} \\
\theta'_{i,k} H + A \geq z_{i,k}^{\min} \\
(1 - \theta'_{i,k}) H + z_{i,k}^{\min} \geq A \\
(1 - \theta'_{i,k}) H + z_{i,k}^{\min} \geq z_{i,k}^{\max} \\
z_{i,k}^{\max} \geq A \\
\theta'_{i,k} H + A \geq z_{i,k}^{\max} \\
z_{i,k}^{\max} \geq z_{i,k}^{\min}
\end{cases}
\end{cases}
\end{cases}
$$

4.9.2.d Linéarisation de la fonction objectif

Calcul du temps de présence par utilisateurs

Les paquets élémentaires calculés pour un nombre infini de machines vont nous permettre le calcul des temps de présence des utilisateurs.

Nous considérerons le début du prochain paquet comme une date échue pour le paquet précédent, on associe à celui-ci un gain positif si le paquet se termine avant le début du prochain paquet et un gain nul sinon. Ainsi, le calcul du temps de présence d'un utilisateur est segmenté par ses paquets et devient alors :

$$T_h = \sum_{paquet_a \in U_h} \min(paquet_{a+1}.min, Makespan(paquet_a)) - paquet_a.min$$

Où $Makespan(paquet_a) = \max_{i=1\ldots a} \max_{\mathsf{op} \in paquet_i} C_{\mathsf{op}}$ et C_{op} est la date de fin de la tâche op.

On connaît alors une borne sur le temps de présence d'un utilisateur ; il suffit de faire la somme des longueurs des intervalles de présence minimale :

$$T_h \geq \sum_{paquet_a \in U_h} paquet_a.max - paquet_a.min \tag{4.9.3}$$

Linéarisation du temps de présence

Nous allons reprendre l'égalité 4.9.3 pour linéariser le calcul du temps de présence des utilisateurs. Tout d'abord, on calcule pour chaque utilisateur les paquets obtenus pour un nombre infini de machines. On trie alors les paquets obtenus d'un utilisateur par ordre croissant.

Pour chaque utilisateur et pour chaque paquet P_p notons W_p la date de fin de la dernière tâche du paquet. On a $W_p = \max_{i \in \mathsf{P}_p} t_i + p_i$ et

$$\forall i \in \mathsf{P}_p, W_p \geq t_i + p_i \tag{4.9.4}$$

Notons $\mathsf{P}_p.min = \min_{i \in \mathsf{P}_p} r_i$, l'intervalle de présence engendré par le paquet qui commence à cette date, notons V_p le temps de présence de cet ensemble de tâches avant le début du prochain paquet. Alors :

$$V_p = \min(W_p, \mathsf{P}_{p+1}.min) \tag{4.9.5}$$

avec la convention pour le dernier paquet $V_p = W_p$.

On a alors :

$$T_h = \sum_{\mathsf{P}_p \subset U_h} V_p - \mathsf{P}_p.min$$

Puisque l'on cherche à minimiser les temps de présence, remarquons qu'il suffit d'imposer

l'égalité 4.9.5 et les inégalités 4.9.4 pour que la minimisation de l'objectif impose à V_p sa valeur.

$$\begin{cases} \forall p, \\ \delta_p \in \{0,1\} \\ \qquad V_p \leq \mathsf{P}_{p+1}.min \\ \qquad V_p \leq W_p \\ \qquad \mathsf{P}_{p+1}.min \leq W_p + \delta_p H \\ \qquad \mathsf{P}_{p+1}.min \geq W_p + (\delta_p - 1)H \\ \qquad V_p \geq W_p + (\delta_p - 1)H \\ \qquad V_p \geq \mathsf{P}_{p+1}.min - \delta_p H \end{cases}$$

avec H suffisamment grand.

Si $\delta = 0$ alors

$$\begin{cases} V_p \leq \mathsf{P}_{p+1}.min \\ V_p \leq W_p \\ \mathsf{P}_{p+1}.min \leq W_p \\ V_p \geq \mathsf{P}_{p+1}.min \end{cases}$$

soit $V_p = \mathsf{P}_{p+1}.min \leq W_p$ et réciproquement si $\mathsf{P}_{p+1}.min \leq W_p$ alors $\delta = 0$.

Si $\delta = 1$ alors

$$\begin{cases} V_p \leq \mathsf{P}_{p+1}.min \\ V_p \leq W_p \\ \mathsf{P}_{p+1}.min \geq W_p \\ V_p \geq W_p \end{cases}$$

soit $V_p = W_p \leq \mathsf{P}_{p+1}.min$ et réciproquement si $W_p \leq \mathsf{P}_{p+1}.min$ alors $\delta = 1$.

Ainsi $V_p = \min(W_p, \mathsf{P}_{p+1}.min)$.

4.9.2.e Nombres de variables

Les données du problème sont les suivantes :

Données du problème :

- M
- r_i
- p_i
- α_i
- T
- K_i
- N^{op}
- Γ
- $NbOpTotal = N^{op}\Gamma$

– H

Pour donner une estimation de la taille du programme linéaire, nous allons énumérer chaque variable. Les variables réelles du problème sont :

Variables réelles :

– $t_{i,k}$: $NbOpTotal$
– T_h : $N \leq N^{op}$
– $y_{i,k}^{\min}$: $NbOpTotal$
– $y_{i,k}^{\max}$: $NbOpTotal$
– $z_{i,k}^{\min}$: $NbOpTotal$
– $z_{i,k}^{\max}$: $NbOpTotal$

Les variables entières du problème sont :

Variables entières :

– $m_{i,k,l}$: $M \times NbOpTotal$
– x_{i_1,k_1,i_2,k_2} : $NbOpTotal^2$
– $Z_{i_1,k_1,i_2,k_2,l}$: $NbOpTotal^2$
– $\eta_{i,k}$: $NbOpTotal$
– $\eta'_{i,k}$: $NbOpTotal$
– $\theta_{i,k}$: $NbOpTotal$
– $\theta'_{i,k}$: $NbOpTotal$
– δ_p : une par paquet $\leq NbOpTotal$

Variables auxiliaires :

Ces variables sont simplement des notations de quantités intermédiaires et sont calculées directement à partir des autres.

– $y_{i_1,k_1,i_2,k_2,l}$: $NbOpTotal^2$
– z_{i_1,k_1,i_2,k_2} : $NbOpTotal^2$

Ce premier modèle contient donc $O(NbOpTotal)$ variables réelles et $O(NbOpTotal^2)$ variables entières pour $O(M \times NbOpTotal^2)$ contraintes.

4.10 Présentation des résultats obtenus grâce au modèle linéaire avec phase transitoire

4.10.1 Format de description d'un problème

Nous proposons le format d'entrée suivant pour décrire un problème périodique :

```
NbUtilisateurs NbMachines
 [ NbTachesUtilisateur
   [  Durée Disponibilité Période ]* ]*
```

Le fichier commence par une ligne qui mentionne le nombre d'utilisateurs et le nombre de machines. Ensuite pour chaque utilisateur une ligne fournit le nombre de tâches suivi d'autant de lignes par tâches, chacune de ces lignes contient la durée, la date de disponibilité et la période de la tâche en question.

4.10.2 Premier problème

Dans ce premier problème, il y a deux utilisateurs dont les opérations sont plus grandes que la période. Ainsi ces utilisateurs sont toujours présents et le critère est constant. Ce problème est utilisé afin de tester le comportement du programme linéaire dans un tel cas.

```
2   4                   // 2 clients et 4 machines
1   4                   // 1 op pour le client 1  periodicite 4
1   10   4              // op 1, duree 10 et dispo 4
1   4                   // 1 op pour le client 2 avec une periodicite de 4
2   6    0              // op 2, duree 6 et dispo 0
```

D'après le théorème 4.9 il est nécessaire de résoudre sur au moins 3 périodes.

Le tableau suivant montre les temps de calcul pour résoudre ce problème selon le critère considéré et selon le nombre de périodes et la période du motif. La résolution n'a pas aboutie pour les cases vides.

	p1n5	p1n6	p1n8	p2n6
Temps de présence total	0m24			101m45
Temps de présence maximal	0m27			
Somme des débits	0m16	1m56		
Débit minimal	1.7s		56m1	

Tab. 4.2 Temps d'exécutions

Le paramètre n indique le nombre de périodes à prendre en compte, sachant qu'on impose que le motif soit complet à la dernière période.

4.10.2.a Temps de présence total, avec $p = 1, n = 5$

La résolution de ce problème a fournit la solution suivante, voir figure 4.10.1 :

```
Tâche < 1 : 1 >  Debut =  6  Machine = 1
Tâche < 1 : 2 >  Debut = 10  Machine = 3
Tâche < 1 : 3 >  Debut = 14  Machine = 2
Tâche < 1 : 4 >  Debut = 18  Machine = 4
Tâche < 1 : 5 >  Debut = 22  Machine = 3
Tâche < 2 : 1 >  Debut =  0  Machine = 1
Tâche < 2 : 2 >  Debut =  4  Machine = 3
Tâche < 2 : 3 >  Debut =  8  Machine = 2
Tâche < 2 : 4 >  Debut = 12  Machine = 4
Tâche < 2 : 5 >  Debut = 16  Machine = 1
```

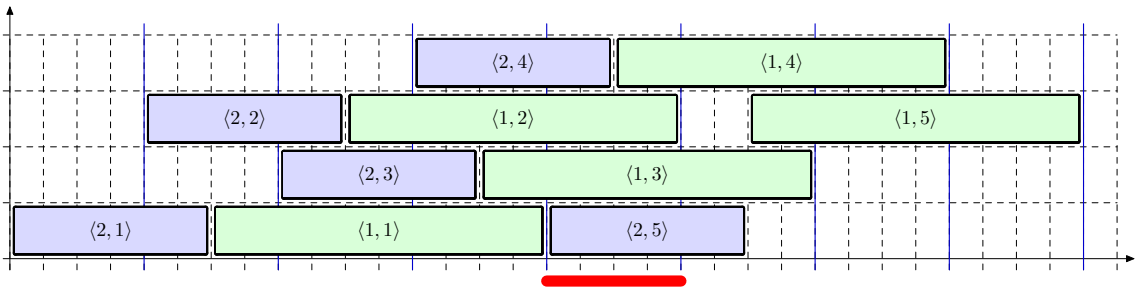


Fig. 4.10.1 Temps de présence total. Le motif est complet dans la période marquée en rouge.

4.10.3 Second problème

Ce problème contient 3 utilisateurs sur 3 machines, afin de montrer la flexibilité du modèle, une contrainte de précédence a été rajoutée entre la tâche $\langle 1, 2\rangle$ et la tâche $\langle 3, 1\rangle$.

```
3   3              // 3 clients et 3 machines
1   3              // 1 op pour le client 1  periodicite 3
1   2   0          // op 1, duree 2 et dispo 0
1   6              // 1 op pour le client 2 avec une periodicite de 6
2   4   0          // op 2, duree 4 et dispo 0
1   6              // 1 op pour le client 3, periodicite 6
3   8   0          // op 3, duree 8 et dispo 0
1                  // Nombre de contraintes supplémentaires (option)
1   2   3   1      // Contrainte <1, 2> -> <3, 1>
```

	p1n2	p1n3	p1n4
Temps de présence total	1s	2m10	72m4
Temps de présence maximal		1s	21s
Somme des débits		28s	4m20
Débit minimal		5m15	7m43

Tab. 4.3 Temps de résolution

4.10.4 WeightedMinMaxPresence, $p = 1, n = 3$

La résolution de ce problème a fourni la solution suivante, voir figure 4.10.2 :

```
Tâche < 1 : 1 >  Debut =  0  Machine = 1
Tâche < 1 : 2 >  Debut =  3  Machine = 3
Tâche < 1 : 3 >  Debut =  6  Machine = 1
Tâche < 1 : 4 >  Debut =  9  Machine = 3
Tâche < 1 : 5 >  Debut = 12  Machine = 1
```

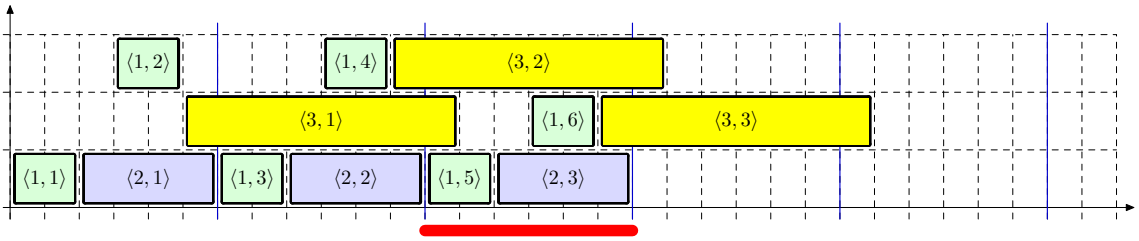


Fig. 4.10.2 Débit minimal

```
Tâche < 1 : 6 >  Debut = 15  Machine = 2

Tâche < 2 : 1 >  Debut =  2  Machine = 1
Tâche < 2 : 2 >  Debut =  8  Machine = 1
Tâche < 2 : 3 >  Debut = 14  Machine = 1

Tâche < 3 : 1 >  Debut =  5  Machine = 2
Tâche < 3 : 2 >  Debut = 11  Machine = 3
Tâche < 3 : 3 >  Debut = 17  Machine = 2
```

4.11 Modèle linéaire avec recherche d'un motif

Afin de réduire la taille du programme linéaire, nous présentons maintenant un autre modèle pour ce problème. Celui-ci est basé uniquement sur les tâches qui constituent le motif.

4.11.1 Remarques préliminaires

On travaille dans ce modèle sur une unique période contenant un motif complet. La phase transitoire est reconstruite à partir du motif trouvé.

La période est de durée T, multiple de toutes les périodes des tâches. Les dates dans le motif sont prises dans l'intervalle $[0, T]$ étant donné qu'on raisonne modulo la période. Une tâche est de durée p_i, de période α_i, de début d'exécution dans le motif $t_{i,k}$ et de date de disponibilité dans le motif $r_{i,k} \equiv r_i + (k-1)\alpha_i \pmod{T}$. Dans le recherche d'un motif, les dates de disponibilités jouent un rôle particulier, en effet d'après le résultat sur le nombre maximum de périodes transitoires une tâche dans le motif qui s'exécute avant sa date de disponibilité dans le motif est une tâche arrivé à la période précédente. Ainsi, les dates de disponibilité interviendront seulement pour le calcul de la fonction objectif.

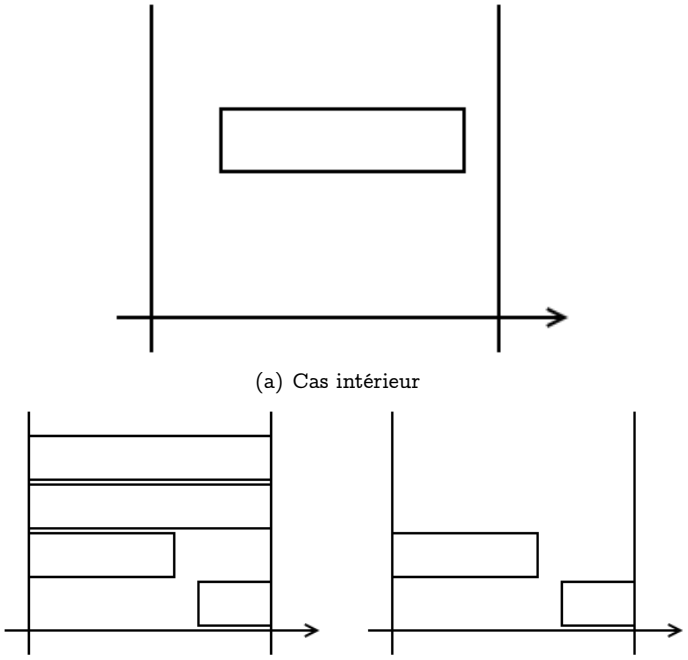
Une tâche dans un motif est constituée de morceaux des tâches dans la période, tels que la tâche est en réalité constituée de la réunion de ces morceaux.

Le nombre de tâches dans le motif est égal au nombre total de tâches reçues par période, soit T/α.

Nous distinguons alors deux cas :

– La tâche est entièrement incluse dans la période. Nous dirons que la tâche est une tâche *intérieure* au motif considéré.

– La tâche est découpée sur la période, avec un certain nombre de morceaux ; ceux-ci occupent une machine entière (éventuellement aucune).



(a) Cas intérieur

(b) Cas découpé

Fig. 4.11.1 .

De plus, dans le cas d'une tâche découpée, celle-ci peut se retrouver sur $\lfloor p/T \rfloor$ ou $\lfloor p/T \rfloor + 1$ périodes (voir la figure 4.11.1). Parmi ces périodes il y en a $\lfloor p/T \rfloor$ ou $\lfloor p/T \rfloor - 1$ pendant lesquelles la tâche est exécutée durant la période entière.

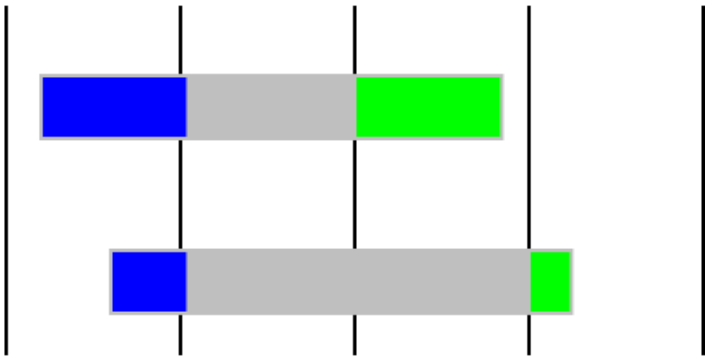


Fig. 4.11.2 Exemples de découpage

Une tâche découpée se compose des 3 morceaux suivants (voir figure 4.11.2) :
– Le début (en bleu sur la figure) se termine à la fin de la période dans le motif.
– La fin (en vert sur la figure) débute au début de la période dans le motif.

– Les parties centrales (en gris sur la figure) consistent en plusieurs tâches (éventuellement aucune) et occupent toute une machine pendant la période dans le motif.

Sur l'exemple de la figure, on peut constater que selon sa date d'exécution, la tâche utilisera soit une machine soit deux dans le motif.

Tâches dans le motif

Soient $i = 1 \dots N$ et $k = 1 \dots K_i$ ($K_i = T/\alpha_i$), alors on a :

$$t_{i,k} \equiv t_i^{k \pmod{K_i}} \pmod{T}$$

et

$$t_{i,k+bK_i} = t_i^k + aT$$

4.11.2 Définition des variables

Une tâche est intérieure si et seulement si elle se termine dans le motif ($t + p \leq T$). Sinon elle découpée.

Variable de découpage

Pour chaque tâche découpée notons par $debut$ et fin, les variables de durée des morceaux de début et de fin de la tâche. Nous avons alors pour une tâche découpée $p_i = debut_{i,j} + a_{i,j}T + fin_{i,j}$ avec $a_{i,j}$ qui est le nombre de périodes complètes occupées par la partie centrale de la tâche i^j. Comme $a_{i,j}$ ne peut prendre que deux valeurs ($a_{i,j} = \lfloor p_i/T \rfloor$ ou $a_{i,j} = \lfloor p_i/T \rfloor - 1$), notons $\delta_{i,j}$ la variable binaire telle que $\delta_{i,j} = \lfloor p/T \rfloor - a_{i,j}$.

$$a_{i,j} + \delta_{i,j} = \lfloor p_i/T \rfloor$$

Exemple : supposons que $p_i = 3.4T$ alors $a_i = 3 - \delta_i$ donc en fonction de α, on aura : $a(i) = 3$ ou $a(i) = 2$.

Notons que si $t_i + p_i \leq T$, alors on a forcément $a_i = 0$ et $\delta_i = 0$. Par convention, si une tâche n'est pas découpée $a_i = 0$ et on prend $debut_i = 0$ et $fin_i = 0$.

On peut se demander s'il Est toujours intéressant de découper une tâche avec a_i minimal. En réalité ? le nombre de découpages dépend de là où commence la période, une tâche peut être plus ou moins découpée selon le motif choisi pour un même ordonnancement.

Variables binaires de placement

– $mach_{i,k}^{debut} = 1$ si le morceau de début de la tâche i dans le motif se trouve sur la machine k, 0 sinon (ou si la tâche n'est pas découpée).
– $mach_{i,k}^{fin} = 1$ si le morceau de fin de la tâche i dans le motif se trouve sur la machine k, 0 sinon (ou si la tâche n'est pas découpée).
– $mach_{i,k}^{interieur} = 1$ si la tâche i dans le motif n'est pas découpée et si elle se trouve sur la machine k, 0 sinon.

Données

Les données du problème sont :
- Il y a $NbOpTotal$.
- p_i : durée de la tâche générique i.
- NbReplicas(i) est le nombre de fois où la tâche générique i est présente dans une période.
- $r_{i,j}$: date de disponibilité de la tâche $\langle i,j \rangle$

Contraintes de découpage

Soit la variable binaire $\beta_{i,j}$ telle que :
- $\beta_{i,j} = 0$ si la tâche $\langle i,j \rangle$ est découpée dans le motif.
- $\beta_{i,j} = 1$ si la tâche $\langle i,j \rangle$ est entièrement incluse dans le motif.

On a donc :

$$\begin{array}{rl} & t_{i,j} + p_i \leq T \quad +(1-\beta_{i,j})H \\ \beta_{i,j}H+ & t_{i,j} + p_i \geq T \end{array}$$

Morceaux de début et de fin

$$\begin{array}{rcl} 0 \leq debut_i \leq T \\ 0 \leq fin_i \leq T \end{array}$$

Contrainte sur les durées des morceaux

$p_i(1-\beta_i) = debut_i + a_i T + fin_i$
On veut aussi que si $\beta_i = 0$ alors $debut_i = T - t_i$ ainsi :

$$\begin{array}{rcl} p_i(1-\beta_i) = debut_i + a_i T + fin_i \\ debut_i \leq T - t_i + \beta_i H \\ debut_i \geq T - t_i - \beta_i H \end{array}$$

Une seule machine par morceau au plus

Pour toutes les machines k et les tâches i, chaque morceau est au plus sur une machine :

$$\begin{array}{l} \forall i, \sum_k mach_{i,k}^{fin} = 1 - \beta_i \\ \forall i, \sum_k mach_{i,k}^{debut} = 1 - \beta_i \end{array}$$

et il ne faut pas plus d'un morceau de début (resp. de fin) par machine :

$$\begin{array}{rcl} \forall k, \sum_i mach_{i,k}^{fin} \leq 1 \\ \forall k, \sum_i mach_{i,k}^{debut} \leq 1 \end{array}$$

Notons que le numéro de machine de la tâche i est $\sum_k mach_{i,k} k$.

Tâches intérieures au motif

Le nombre de machines occupées par les morceaux centraux des tâches découpées vérifie :

$$\forall i, \sum_{k=1}^{M} mach_{i,k}^{interieur} = \beta_i$$

Machines disponibles

Réordonnons les machines du motif de sorte que les parties centrales des tâches soient toutes sur les premières machines. Ainsi, les machines de 1 à $\sum a_i$ sont prises par les parties centrales des tâches. Il reste $M - \sum a_i$ machines.

Soient les variables binaires γ_k telles que $\gamma_k = 1$ si la machine k n'est pas occupée par une partie centrale, et 0 sinon.

Les machines de 1 à $\sum_{i=1} a_i$ sont occupées, ce qui peut s'écrire :

$$\begin{array}{rcl} \forall k, \gamma_{k+1} & \geq & \gamma_k \\ M & = & \sum_i a_i + \sum_k \gamma_k \end{array}$$

Comme $a_i \geq \lfloor p_i/T \rfloor - 1$, on sait aussi que les premières machines de 1 à $\sum_i \lfloor p_i/T \rfloor - 1$ sont certainement occupées :

$$\forall k \in \{1, \ldots, \sum_i \lfloor p_i/T \rfloor - 1\}, \gamma_k = 0$$

$\gamma_{k+1} \geq \gamma_k$ exprime le fait que si la machine k n'est pas occupée alors la suivante ne l'est pas non plus.

$\sum_k \gamma_k = \sum_i a_i$ impose qu'il y a autant de machines occupées que de morceaux de tâches occupant une machine.

Dès le départ on sait qu'au moins $\sum_{p_i \geq T} (\lfloor p_i/T \rfloor - 1)$ machines sont occupées. Ainsi, il est possible de déclarer les $\gamma_k, k = 1 \ldots \sum_{p_i \geq T} (\lfloor p_i/T \rfloor - 1)$ à 0.

De plus on doit exiger qu'une machine occupée par une partie centrale n'est pas utilisée par d'autres tâches. Ainsi, on doit avoir pour toutes les variables de placement :

$$\begin{array}{rcl} mach_{i,k}^{debut} & \leq & \gamma_k \\ mach_{i,k}^{fin} & \leq & \gamma_k \\ mach_{i,k}^{interieur} & \leq & \gamma_k \end{array}$$

Disjonction avec les morceaux de début ou de fin

On retrouve des contraintes analogues au modèle précédent. Les tâches sur une même machine ne se chevauchent pas :

$$\begin{array}{lrcl} \forall i_1, j_1, i_2, j_2, k, & fin_{i_1,j_1} & \leq & t_{i_2,j_2} + (2 - mach_{i_1,j_1,k}^{fin} - mach_{i_2,j_2,k}^{int})H \\ \forall i_2, j_2, i_3, j_3, k, & t_{i_2,j_2} + p_{i_2} + debut_{i_3,j_3} & \leq & T + (2 - mach_{i_3,j_3,k}^{debut} - mach_{i_2,j_2,k}^{int})H \\ \forall i_1, j_1, i_3, j_3, k, & fin_{i_1,j_1} + debut_{i_3,j_3} & \leq & T + (2 - mach_{i_3,j_3,k}^{debut} - mach_{i_1,j_1,k}^{fin})H \end{array}$$

Exclusion des morceaux entre eux

$$\begin{array}{lrcl}
\forall (i_1,j_1) \neq (i_2,j_2), & x_{i_1,j_1,i_2,j_2} + x_{i_2,j_2,i_1,j_1} & = & 1 \\
\forall (i_1,j_1) \neq (i_2,j_2), & t_{i_2,j_2} & \leq & t_{i_1,j_1} + H x_{i_1,j_1,i_2,j_2} \\
\forall (i_1,j_1) \neq (i_2,j_2), & k, t_{i_1,j_1} + p_{i_1} & \leq & t_{i_2,j_2} + H(3 - x_{i_1,j_1,i_2,j_2} - mach^{interieur}_{i_1,j_1,k} - mach^{interieur}_{i_2,j_2,k})
\end{array}$$

Linéarisation de la fonction objectif

Pour une tâche fixée avec sa date de disponibilité, nous ajustons la date de début d'exécution : si la date de début se trouve avant la disponibilité de la période, alors la date de début est avancée d'une période T. Ceci permet de conserver l'ordre des dates de disponibilités dans le calcul de l'objectif.

Si $t \geq r$ alors $t' = t$.

Si $t < r$ alors $t' = t + T$.

Soit :

$$\begin{array}{lcll}
t'_{i,j} & = & t_{i,j} + \theta_{i,j} T, \theta_{i,j} \in 0,1 \\
t'_{i,j} & \geq & r_{i,j}
\end{array}$$

θ est libre si $t \geq r$ mais étant donné qu'on cherche à minimiser les temps de présence, il sera calé à 0 dans ce cas.

Calcul des intervalles de présence des utilisateurs $[r, t']$

Les dates de disponibilités étant fixées, le calcul des temps de présence pour chaque utilisateur revient à une somme de maximums pris entre la disponibilité suivante et la fin de la tâche actuelle.

Une tâche i contribue au temps de présence par l'intervalle $[r_i, t'_i + p_i]$. Il faut se ramener dans une période : si $t'_i + p_i \leq T$ alors c'est inchangé, sinon $[0, \min T, t'_i + p_i - T] \cup [r_i, T]$.

On suppose les intervalles triées par disponibilité croissante :

$$\begin{array}{lrcl}
\forall i,j, & t'endmax_{i,j} & \geq & t'end_{i,j} \\
\forall i,j, & (i,j) \in U_u, t'endmax_{i,j} & \geq & t'max^u \\
\forall i,j, & (i,j) \in U_u, t'max^u & \geq & fin_{i,j} \\
\forall i,j, & (i,j) \in U_u, t'max^u & \geq & t'end_{i,j} - T
\end{array}$$

$t'max$ sera forcément calée au maximum des dates de fin des tâches de l'utilisateur à cause de l'objectif, en prenant compte aussi les morceaux de fin. Ensuite les dates de fin des intervalles sont triées, si la date de disponibilité de (i,j) est immédiatement suivie de (i',j'), on a :

$$\begin{array}{rcl}
t'endmax_{i,j} & \leq & t'endmax_{i',j'} \\
t'endmax_{i',j'} & \geq & t'end_{i',j'} \\
s_{i,j} & = & \min(r_{i',j'}, t'endmax_{i,j})
\end{array}$$

Notons aussi que si un utilisateur a une tâche de durée supérieure à la période, on sait que son temps de présence sera égal à la période. Ce qui permettra d'éviter le calcul complet pour cet utilisateur. De même fe façon générale lorsqu'un utilisateur est permanent.

– Si la tâche est intérieure on a $\beta_i = 1$ et $debut_i = fin_i = a_i = 0$.

- Si $\theta_i = 0$, l'intervalle est $[r_i, t_i + p_i]$
- Si $\theta_i = 1$, l'intervalle est $[r_i, T] \cup [0, t_i + p_i]$

– Si la tâche est découpée alors $\beta_i = 0$ et $t_i + p_i \geq T$.
- Si $a_i \geq 1$ ou $\theta_i = 1$, alors l'intervalle est $[0, T]$.
- Si $a_i = 0$ et $\theta_i = 0$ alors $[0, fin_i] \cup [r_i, T]$ Notons que dans ce cas $fin_i = p_i - debut_i = p_i + t_i - T$

Distinguons les intervalles qui commencent à la date 0.
Posons pour chaque tâche i d'un utilisateur u donné :

$$\begin{array}{rcl} s_0^u & = & \min(r_{\text{1ere release de } U_u}, t'max^u) \\ s_0'^u & = & \max(0, \min(T, \max_i(t'_i + p_i - T))) \\ TP^u & = & \sum_{(i,j)\in U_u}(s_i - r_i) \end{array}$$

Variables

Récapitulons les variables employées pour ce problème :

$$\begin{array}{rcl} u & \in & \{User_1, User_2, \ldots\} \\ s_0^u & \geq & 0 \\ s_0'^u & \geq & 0 \end{array}$$

$t'max^u$ libre

$$\begin{array}{rcl} i & \in & \{1, \ldots, NbOpTotal\} \\ j & \in & \{1, \ldots, NbReplicas(i)\} \\ t_{i,j} & \geq & 0 \\ t'_{i,j} & \geq & 0 \\ t'end_{i,j} & \geq & 0 \\ t'endmax_{i,j} & \geq & 0 \\ s_{i,j} & \geq & 0 \\ s'_{i,j} & \geq & 0 \\ debut_{i,j} & \geq & 0 \\ fin_{i,j} & \geq & 0 \\ a_{i,j} & \geq & 0 \end{array}$$

Variables binaires

$\beta_{i,j}$
$\delta_{i,j}$
$\theta_{i,j}$
$\gamma_{i,j}$

$k \in \{1, \ldots, M\}$
$mach_{i,k}^{debut}$
$mach_{i,k}^{fin}$
$mach_{i,k}^{interieur}$

$$
\begin{array}{rcl}
i_1 & \in & \{1,\ldots,NbOpTotal\} \\
j_1 & \in & \{1,\ldots,NbReplicas(i)\} \\
i_2 & \in & \{1,\ldots,NbOpTotal\} \\
j_2 & \in & \{1,\ldots,NbReplicas(i)\}
\end{array}
$$

x_{i_1,j_1,i_2,j_2}

Variables entières

$a_{i,j} \geq 0$

4.11.2.a Variables auxiliaires

Soit $\bar{t}_{i,j}$ la date d'exécution de la tâche $\langle i,j \rangle$ ramenée sur une seule machine.
$machine_{i,j}$ désigne le numéro de la machine où s'exécute la tâche $\langle i,j \rangle$.
$t'_{i,j}$ désigne la date d'exécution réelle de la tâche $\langle i,j \rangle$, avec la variable binaire $\theta_{i,j}$ de décalage.
$t'^{end}_{i,j}$ désigne la date de fin d'exécution de la tâche $\langle i,j \rangle$.
Les variables binaires de précédence x_{i_1,j_1,i_2,j_2} pour $(i_1,j_1) \neq (i_2,j_2)$.
$maxend_{k,u}$, $maxend'_{k,u}$ est la date de fin des k premières tâches de l'utilisateur u.

4.11.2.b Contraintes

La date d'exécution $t_{i,j}$ dans le motif est alors :

$$
\begin{array}{rcl}
t_{i,j} & \geq & 0 \\
t_{i,j} & \leq & MT \\
t_{i,j} & \equiv & \bar{t}_{i,j} \pmod{T}
\end{array}
$$

Soit écrit de façon linéaire :

$$
\begin{array}{rcl}
t_{i,j} & = & \bar{t}_{i,j} - (machine_{i,j} - 1)T \\
(machine_{i,j} - 1)T & \leq & \bar{t}_{i,j} \\
\bar{t}_{i,j} & \leq & machine_{i,j}T
\end{array}
$$

Le motif suivant doit pouvoir succéder au motif actuel.

$$\forall i_1,j_1,i_2,j_2, \bar{t}_{i_1,j_1} + MT \geq \bar{t}_{i_2,j_2} + p_{i_2,j_2}$$

Contraintes de disjonction sur la machine :

$$
\begin{array}{lrcl}
\forall (i_1,j_1) \neq (i_2,j_2), & x_{i_1,j_1,i_2,j_2} + x_{i_2,j_2,i_1,j_1} & = & 1 \\
\forall (i_1,j_1) \neq (i_2,j_2), & \bar{t}_{i_1,j_1} + p_{i_1} & \leq & \bar{t}_{i_2,j_2} + Hx_{i_2,j_2,i_1,j_1}
\end{array}
$$

Décalage des tâches dans le motif :

$$
\begin{array}{rcl}
t'_{i,j} & = & t_{i,j} + \theta_{i,j}T, \theta_{i,j} \in 0,1 \\
t'_{i,j} & \geq & r_{i,j}
\end{array}
$$

Linéarisation de l'objectif

$$t'^{end}_{i,j} = t'_{i,j} + p_{i,j}$$

Notons que si l'utilisateur a une tâche de durée supérieure à la période alors il est forcément présent pendant toute la durée d'une période et son temps de présence est T. Nous calculerons donc le temps de présence des utilisateurs n'ayant pas de tâches de durée supérieure à la période.

Commençons par calculer la longueur (noté $maxend_{0,u}$) de l'intervalle de présence dans le motif dû aux tâches des périodes précédentes. On suppose que l'utilisateur n'a que des tâches de durée inférieure à la période, ces tâches proviennent uniquement du motif précédent.

$$\begin{array}{rrcl} \forall u, & maxend_{0,u} & \geq & 0 \\ \forall i,j,u, & maxend_{0,u} & \geq & t'_{i,j} + p_i - T \end{array}$$

Ces variables $maxend_{0,u}$ seront calées au maximum des $t'_{i,j} + p_i - T$ et de 0 à cause du critère à minimiser. On tri les dates de disponibilité de l'utilisateur u de sorte que $r_{id[k],replica[k]}$ soit triée.

$$\begin{array}{rrcl} \forall k = 1 \dots N_u, & maxend_{k,u} & = & \min(t'^{end}_{id[k],replica[k]}, T) \\ \forall k = 1 \dots N_u, & maxend'_{k,u} & \geq & maxend'_{k-1,u} \\ \forall k = 0 \dots N_u, & maxend'_{k,u} & \geq & maxend_{k-1,u} \end{array}$$

$$\begin{array}{rrcl} \forall k = 0 \dots N_u - 1, & minmaxend'_{k,u} & = & \min(maxend'_{k,u}, r_{id[k],replica[k]}) \\ & minmaxend'_{N_u,u} & = & maxend'_{N_u,u} \end{array}$$

$$TP_u = \sum_{k \in u} minmaxend'_{k,u} - \sum_{\langle i,j \rangle \in u} r_{i,j}$$

4.11.2.c Nombres de variables

Données du problème :
- T
- α_i pour chaque $i = 1 \dots N$
- p_i pour chaque $i = 1 \dots N$
- $r_i, r_{i,j}$
- Variables auxiliaires :
  - $K_i = T/\alpha_i$

Variables réelles :
- $t_{i,k}$ pour chaque $i = 1 \dots N$ et $k = 1 \dots K_i$
- $debut_{i,k}$
- $fin_{i,k}$
- $t'_{i,j}$
- $t'endmax_{i,j}$
- $t'end_{i,j}$

- $t'max^u$
- s_0^u

Variables entières :

- $a_{i,j}$ nombre de périodes complètes occupées par la partie centrale de la tâche i^j.
- x_{i_1,j_1,i_2,j_2}
- $\delta_{i,j}$
- $mach_{i,k}^{debut}$
- $mach_{i,k}^{fin}$
- $mach_{i,k}^{interieur}$
- $\beta_{i,j}$ est un booléen qui indique si (i,j) est découpée dans le motif
- γ_k pour $k = 1,\ldots,M$, ce sont des booléens qui indiquent si la machine k est occupée par une partie centrale
- $\theta_{i,j}$ est un booléen de décalage des tâches d'une période selon leur date de disponibilité

Ce second modèle basé sur le motif contient alors :

- Variables réelles : $O(N)$
- Variables binaires : $O(N^2+U)$
- Contraintes : $O(M*N^2+U)$

4.12 Présentation des résultats obtenus grâce au modèle linéaire avec recherche du motif

La taille du modèle linéaire est réduite en comparaison de la taille du modèle avec prise en compte de la phase transitoire. Ainsi, la durée de résolution des problèmes cités, qui pouvait aller jusqu'à plusieurs minutes, ne dure avec ce nouveau modèle que quelques dixièmes de secondes. Devant ce résultat encourageant, nous avons considéré des problèmes de plus grande taille.

4.12.1 Problème 3

Ce problème contient 4 utilisateurs qui se répartissent 4 machines. Les tâches d'un utilisateur arrivent regroupées ; une succession de tâches arrivent pendant un temps assez court. Chaque utilisateur possède 4 tâches. Le problème est le suivant :

```
4  4
4
   1  54   0  180
   2  34  10  180
   3  61  20  180
   4   2  30  180
4
   5   9  40  180
   6  15  50  180
   7  89  60  180
   8  70  70  180
4
   9  38   80  180
```

Tab. 4.4 Temps de résolution pour le problème 3

Critère	Temps de résolution
Temps de présence total	1.09 sec
Temps de présence maximal	0.92 sec
Somme des débits	0.97 sec
Débit minimal	1.02 sec

```
  10   19    90   180
  11   28   100   180
  12   87   110   180
4
  13   95   120   180
  14   34   130   180
  15    7   140   180
  16   29   150   180
```

4.12.2 Exemple de problème non résolu avec cette méthode

Le problème suivant est en nombre d'opérations de taille comparable à d'autres problèmes résolus, mais la résolution avec CPLEX n'a pas abouti après plusieurs jours de calcul.

```
4  8   // 4 utilisateurs et 8 machines
4
   1    5  0 120
   2   70  0 120
   3   45  0 120
   4   83  0 120
4
   5   24  0 120
   6   80  0 120
   7   58  0 120
   8   45  0 120
4
   9   29  0 120
  10   56  0 120
  11   29  0 120
  12   61  0 120
4
  13   43 0 120
  14   64 0 120
  15   45 0 120
  16   74 0 120
```

Nous avons décrit dans ce chapitre un problème d'ordonnancement équitable de tâches périodiques. Deux programmes linéaires ont été exposés. Cependant, lorsque le nombre de tâches augmente, il devient nécessaire de proposer une heuristique de calcul de solutions de bonne qualité. Dans le chapitre suivant, nous allons décrire une méthode d'exploration de l'espace des solutions.

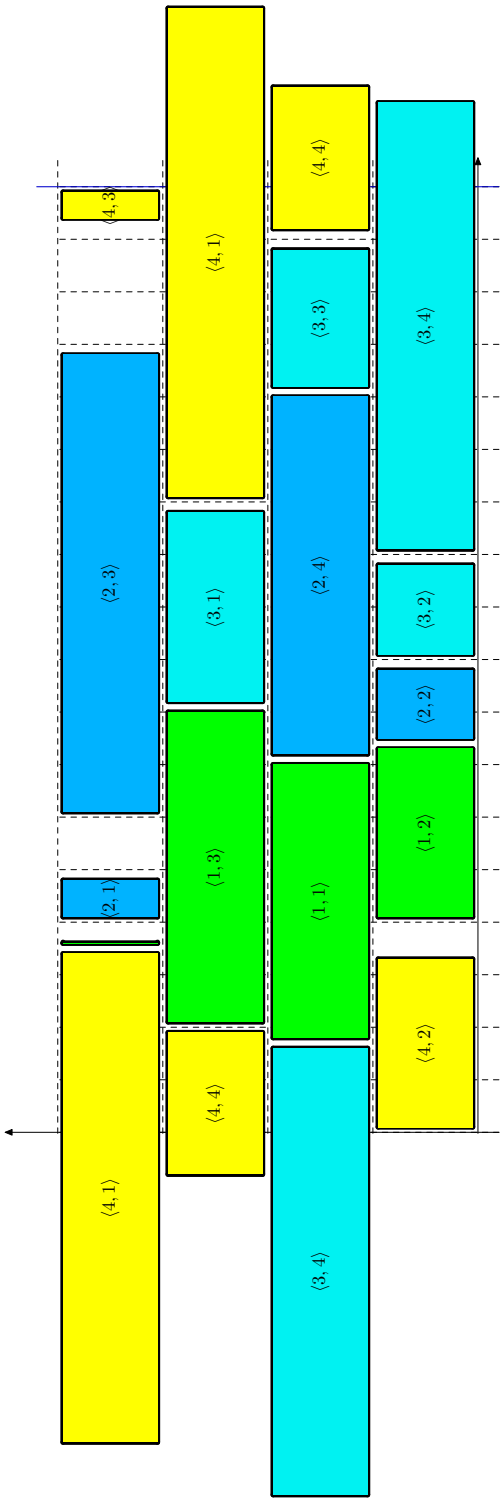


Fig. 4.12.1 Problème 3, Temps de présence total : résolu sous CPLEX en 1.09 secondes

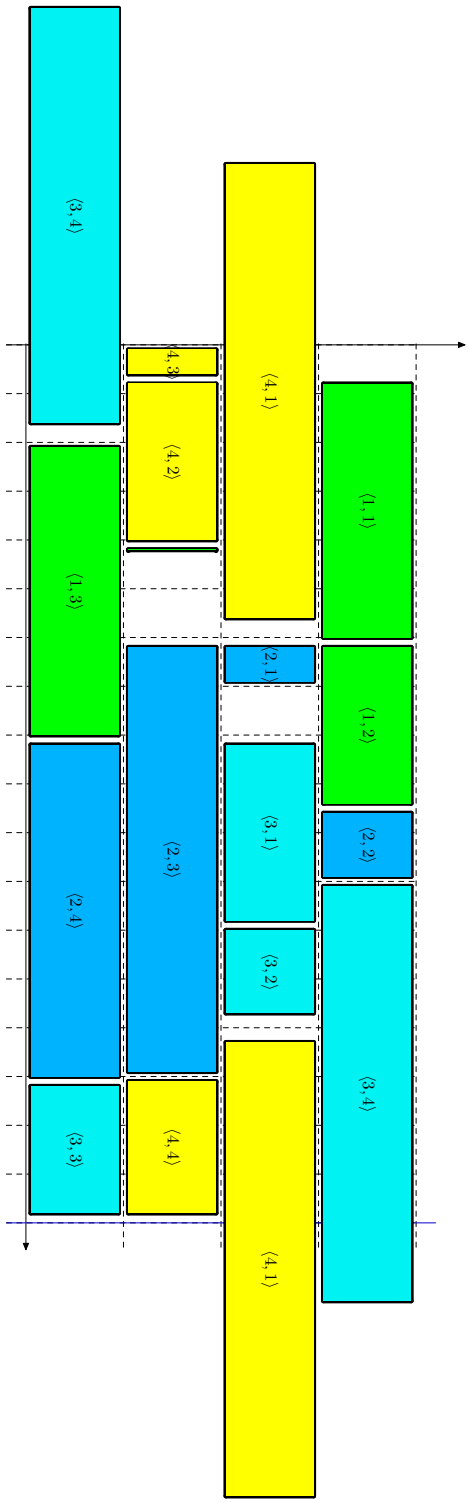


Fig. 4.12.2 Problème 3, Temps de présence maximal : résolu sous CPLEX en 0.97 secondes

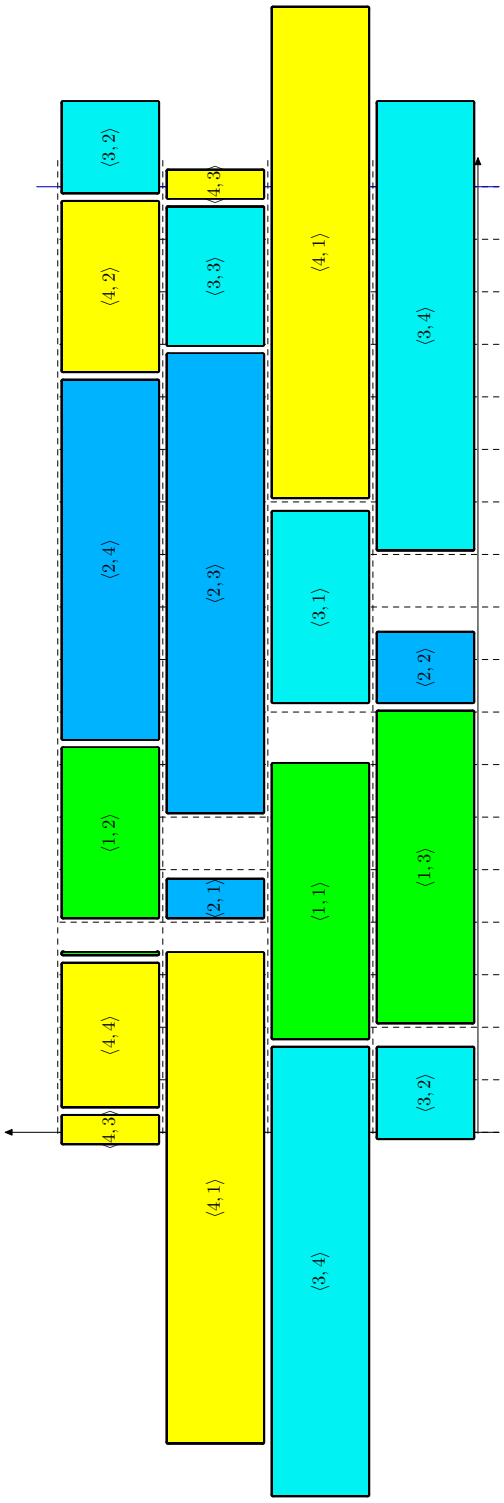


Fig. 4.12.3 Problème 3, Somme des débits : résolu sous CPLEX en 0.92 secondes

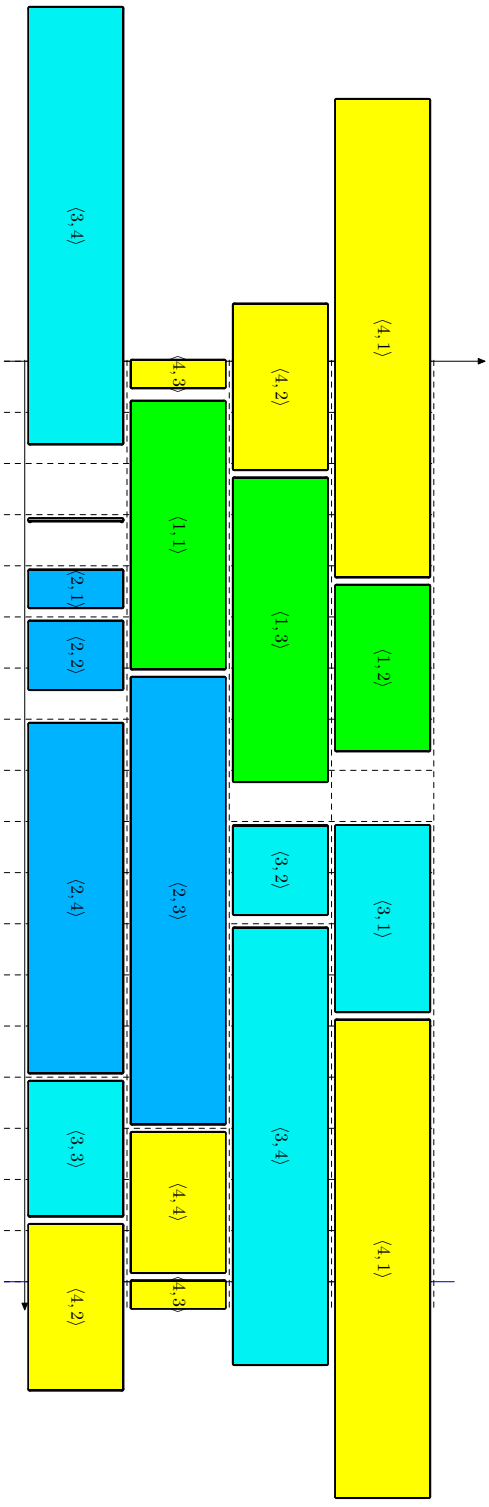


Fig. 4.12.4 Problème 3, Débit minimal : résolu sous CPLEX en 1.02 secondes

Chapitre 5

Résolution heuristique du problème périodique

Heuristique : art d'inventer, de faire des découvertes.

Émile Littré : Dictionnaire de la langue française

Parmi les problèmes d'optimisation combinatoire les plus complexes réside la classe des problèmes NP-difficile [Garey et Johnson, 1990]. Etant donné que l'espace des solutions croît exponentiellement avec la taille du problème, il est souvent infaisable en pratique d'en énumérer toutes les solutions pour en sélectionner la meilleure. La résolution de ces problèmes demande un temps de calcul qui augmente plus rapidement que toutes puissances de la taille du problème. L'étude de ces problèmes a amené à l'élaboration de méthodes de résolution approximative appelées des *heuristiques* [Reeves, 1995; Rayward-Smith et al., 1996; Glover et Kochenberger, 2003]. L'objectif de ces heuristiques est de trouver des solutions de qualité acceptable en un temps raisonnable. L'efficacité d'une heuristique réside dans sa capacité de s'adapter aux instances générales du problème, d'éviter le piège des minimums locaux et d'exploiter certaines propriétés du problème (telles que des symétries ou des règles de dominances).

Ainsi dans le chapitre précédent nous avons établi un modèle linéaire de notre problème, ce modèle fournit une solution optimale cependant lorsque la taille du problème augmente les temps de résolution deviennent irréalistes. Nous avons dû élaborer une approche heuristique plus rapide pour obtenir sur des instances de plus grande taille des solutions de qualité acceptables, mais pas nécessairement optimales.

5.1 Présentation générale de la méthode

Tout d'abord nous allons décrire la terminologie employée :

Définition 5.1 (Espace des solutions)**.** *L'espace des solutions est l'ensemble des instances réalisables du problème (instances qui vérifient un ensemble de contraintes liées au problème considéré). On dit aussi qu'une instance réalisable est une solution admissible.*

Définition 5.2 (Fonction objectif, espace des critères et ordre de comparaison)**.** *La qualité d'une instance est évaluée par une fonction objectif f qui associe à chaque instance x un élément y de l'ensemble des critères. Cet ensemble est pourvu d'un ordre partiel $\prec$ appelé ordre de comparaison. Notons que cet ordre de comparaison s'étend naturellement à l'ensemble des solutions via la fonction objectif f.*

Un problème d'optimisation consiste à chercher dans l'espace des solutions les meilleures instances relatives à l'ordre de comparaison.

Définition 5.3 (Espace des configurations, fonction de codage). *L'espace des configurations est un ensemble abstrait où est associé à chaque élément une instance de l'espace des solutions. La fonction correspondante est appelée fonction de codage. Lorsque cette fonction est surjective sur l'ensemble des solutions, on dit que le codage est complet, sinon on dit que le codage est incomplet.*

Le rôle de la fonction de codage est de simplifier la représentation des solutions du problème en vue de les traiter. Pour ce faire, le codage doit si possible respecter des propriétés de régularité par rapport à l'ordre de comparaison. Ces propriétés facilitent alors la recherche d'une solution de bonne qualité.

Pour notre problème, nous proposons un codage complet d'une solution qui consiste en une permutation des tâches, laquelle définit un ordre entre tâches, et un vecteur de booléens, un booléen par tâche, il indique quand la tâche est calée sur sa date de disponibilité ou non. Ce codage des motifs est complet, on pourra alors lui associer un diagramme de Gantt.

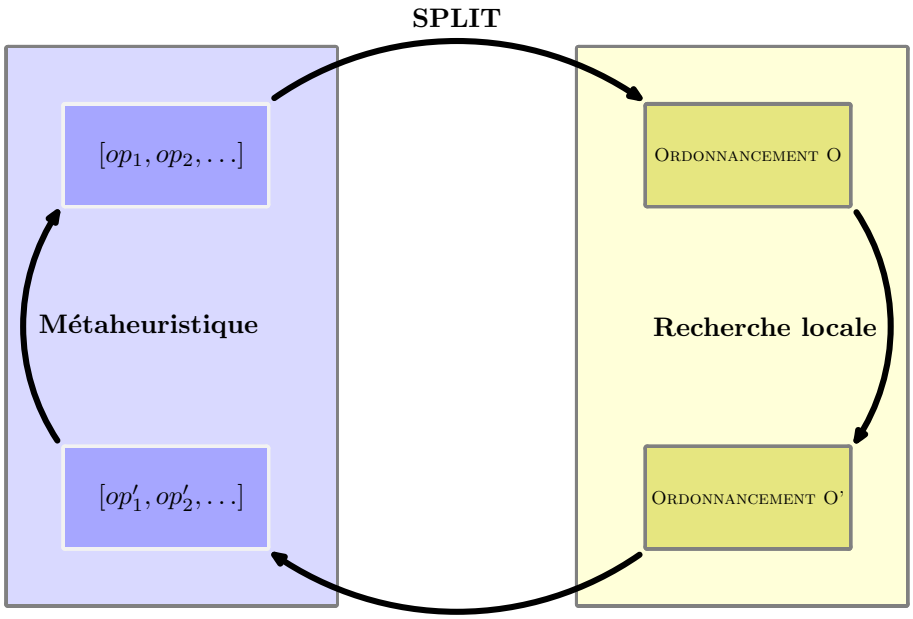


Fig. 5.1.1 Principe général

Le principe général de notre heuristique consiste à utiliser un codage partiel [1] constitué seulement de la permutation des tâches. Ce codage devient complet si on lui ajoute un vecteur de booléens. L'idée est alors, pour un ordre fixé des tâches, de rechercher le meilleur vecteurs de booléens : c'est le rôle de la procédure SPLIT, celle-ci fait correspondre une solution optimale à un ordre fixé entre les tâches.

Afin de limiter les temps de calcul, nous utilisons une procédure LIMITED-SPLIT basée sur SPLIT mais qui impose une limite au nombre de solutions explorées. Cette limitation dans l'exploration des solutions entraîne que la procédure LIMITED-SPLIT n'est pas exhaustive dans l'espace des solutions et donc que les solutions fournies par LIMITED-SPLIT sont sous-optimales.

1. Voir [Duhamel et al., 2011] sur l'idée de résolution par représentation partielle

Nous compléterons alors par une recherche locale qui s'effectue directement dans l'espace des solutions (voir la figure 5.1.1) afin d'utiliser quelques propriétés des critères étudiés.

5.1.1 Approche de résolution pour le cas périodique

Nous avons essayé de généraliser l'idée de l'algorithme du SPLIT aux problèmes d'ordonnancement de tâches à arrivée périodique.

Nous avons observé qu'un ordonnancement périodique se compose de cycles. Chaque cycle s'exécute en boucle sur un sous-ensemble de machines : chaque machine exécute le cycle avec un décalage de période. Une première idée a été de considérer chacun de ces cycles comme une tournée individuelle et de chercher pour un ordre imposé quel est le meilleur découpage. Nous nous sommes heurté au problème suivant : étant donné que les ordonnancements semi-actifs sont dominants, chaque tâche est soit calée sur sa date de disponibilité, soit calée sur la date de fin d'une tâche précédente. Il y a au pire 2^n façons de placer n tâches périodiques qui se suivent sur un cycle. Par conséquent, à chaque arc de i vers j du graphe de l'algorithme, il aurait fallu ajouter à chaque solution de i les 2^n cycles possibles. Mais ceci multiplie le nombre de solutions à chaque arc et rend impraticable l'algorithme.

Nous allons donc présenter une autre idée basée sur les *séquences* :

Définition 5.4 (Séquence)**.** *Une séquence est une suite de tâches dont la première est calée à sa date de disponibilité et les suivantes sont calées sur la date de fin de la tâche précédente.*

Puisque les machines sont identiques, il importe peu de savoir sur quelles machines sont placées les tâches d'une séquence. Il n'y a alors qu'une façon de placer les tâches dans une séquence (modulo le placement sur les machines). Nous considérons ensuite qu'un ordonnancement est composé d'une suite de séquences, vues comme des intervalles, sous la contrainte qu'au plus M intervalles se rencontrent en un même point. Ceci signifie que l'ordonnancement correspondant utilise moins de M machines.

L'autre avantage est qu'on a la propriété de séparabilité sur les séquences par rapport aux critères des intervalles de présence : pour calculer les intervalles de présence d'un ordonnancement auquel on rajoute une séquence, il suffit de faire l'union des intervalles de présence de l'ordonnancement avec ceux de la séquence.

Nous découpons alors une "séquence géante" en une suite de séquences via un algorithme inspiré par le SPLIT. Celui-ci va fournir un découpage optimal à partir de cette séquence géante. Nous améliorerons ensuite cette solution via des recherches locales, ce qui nous fournira une nouvelle séquence géante, etc.

5.1.2 Les problèmes de tournées de véhicules

Avant de présenter notre approche nous effectuons ici un rappel sur les problèmes de tournées de véhicules et de la technique du SPLIT dont nous nous sommes inspirés pour notre heuristique. Les problèmes de tournées de véhicules est un problème important dans le domaine de la logistique et des transports [Dantzig et Ramser, 1959]. Ils consistent à construire un ensemble de tournées pour une flotte de m véhicules à disposition, cette flotte devant approvisionner n clients répartis géographiquement. L'objectif est de minimiser le coût total. Chaque véhicule va charger les cargaisons des clients à partir du dépôt central, il va ensuite les acheminer puis les livrer chez les clients

pour finalement revenir au dépôt central. Chaque tournée commence et se termine à un dépôt et doit respecter des contraintes de capacité. La matrice des coûts de transport entre chaque client et entre les clients et le dépôt central est connue. Chaque client U_j a un chargement de poids P_j. On se place dans le cas où tous les véhicules ont la même capacité de chargement C à respecter.

5.1.3 Algorithme du SPLIT appliqué au CVRP

Une tournée géante consiste en un parcours de tous les clients. La tournée géante spécifie quel est le client qui succède à chaque client, tout en parcourant tous les clients une seule fois. La tournée géante spécifie donc un chemin hamiltonien entre les clients. Une tournée géante est notée $\lambda = (U_{\lambda_1} U_{\lambda_2} \dots U_{\lambda_N})$.

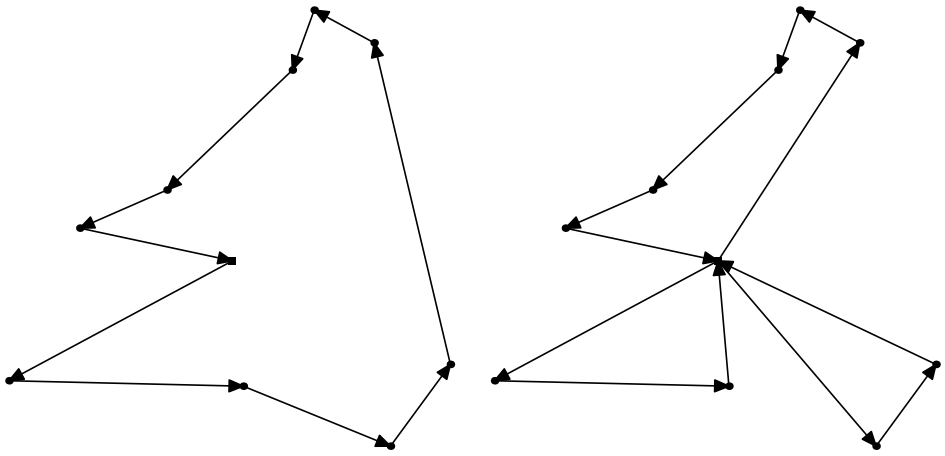


Fig. 5.1.2 Exemple de tournée géante et un découpage de cette tournée avec 3 véhicules.

Une tournée respecte la tournée géante si chaque véhicule visite le client suivant spécifié par la tournée géante ou retourne au dépôt central. Une tournée qui respecte une tournée géante peut se définir comme un *découpage* de la tournée géante. Il suffit de connaître les clients pour lesquels le véhicule va retourner ensuite au dépôt central pour reconstituer une tournée.

L'algorithme du SPLIT [Beasley, 1983; D. M. Ryan et Glover, 1993] consiste à trouver la meilleure tournée de véhicules sous la contrainte d'une tournée géante imposée. Cet algorithme se restreint à explorer seulement l'ensemble des tournées qui respectent la tournée géante.

Description de l'algorithme du SPLIT

L'algorithme procéde de la façon suivante : notons $\lambda^j = (op_{\sigma(1)}, \dots, op_{\sigma(j)})$ le j^e préfixe de la tournée géante $(op_{\sigma(1)}, \dots, op_{\sigma(N)})$. L'algorithme du SPLIT est un algorithme de programmation dynamique sur ces λ^j.

L'idée est de construire successivement les tournées optimales sur les λ^j, en remarquant que le coût total a la propriété de séparation, c'est à dire qu'il est égal à la somme des coûts des tournées individuelles de chaque véhicule. Cela revient à calculer le chemin de coût minimal dans le graphe des N clients auquel on rajoute un sommet source 0, partant du point 0 vers la destination $\sigma(N)$. Il existe alors au plus $\frac{N(N+1)}{2}$ arcs qui sont $\forall 0 \leq a < b \leq N, (a, b)$ et ont pour coût K_a^b le coût de

la sous-tournée $\lambda_a^b = (op_{\sigma(a+1)}, \ldots, op_{\sigma(b)})$ lorsque celle-ci est faisable en respectant les contraintes de capacité.

On passe du nœud a au nœud b pour $a < b$ en ajoutant la tournée individuelle λ_a^b de coût K_a^b. On associe alors à chaque préfixe λ^j du tour géant un nœud j et à chaque tournée individuelle $(op_{\sigma(a+1)}, \ldots, op_{\sigma(b)})$ un arc du nœud a vers le nœud b de coût K_a^b. L'algorithme calcule alors le chemin de coût minimal entre le nœud 0 et le nœud N, ceci va correspondre à un découpage optimal de la tournée géante.

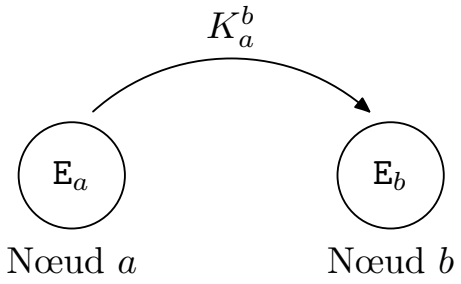


Fig. 5.1.3 Étape $a \to b$

Au début de l'étape a, chaque nœud $j \leq a$ contient un découpage optimal du tour $\lambda^j = (op_{\sigma(1)}, \ldots, op_{\sigma(j)})$ préfixe du tour géant $(op_{\sigma(1)}, \ldots, op_{\sigma(N)})$.

Pour calculer le chemin de coût minimal, on effectue itérativement à l'étape $a = 0, \ldots, N-1$ l'ajout à partir du nœud a au nœud b pour $b = a+1, \ldots, N$ des sous-tournées de la forme $(op_{\sigma(a+1)}, \ldots, op_{\sigma(b)})$. A chaque ajout on garde seulement le découpage de meilleur coût dans le nœud destination.

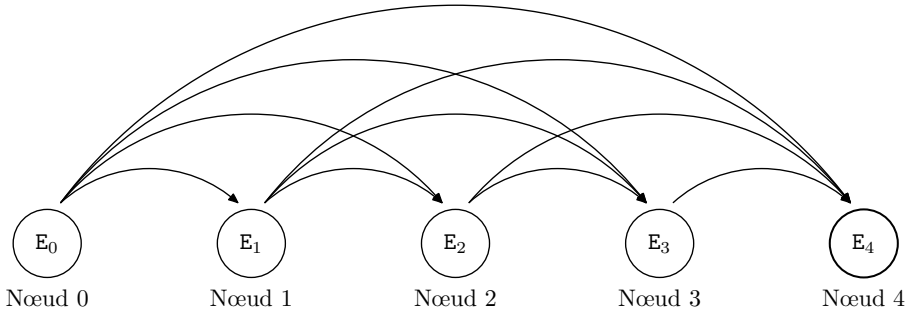


Fig. 5.1.4 Étapes complètes du SPLIT pour 4 tâches.

Dans le cas du CVRP, les labels des nœuds sont de la forme Label = [j, coût] où j indique le nœud précédent et le coût indique le coût optimal trouvé jusqu'à maintenant sur ce label.

Théorème 5.1 (SPLIT-VRP). *L'algorithme SPLIT-VRP fournit le meilleur découpage du tour géant* $(\lambda_1, \ldots, \lambda_N)$ *dans le dernier nœud.*

Démonstration. Par induction, supposons qu'àu début de l'étape k les nœuds λ_i pour $i = 1$ à $k-1$ contiennent chacun le meilleur découpage du tour $(\lambda_1, \ldots, \lambda_i)$. Notons que le meilleur découpage du tour $(\lambda_1, \ldots, \lambda_k)$ consiste en un découpage de $(\lambda_1, \ldots, \lambda_i), i \in [1, k-1]$ et un tour $(\lambda_{i+1}, \ldots, \lambda_k)$. Etant donné que c'est le meilleur découpage possible, il faut que le découpage de

Listing 5.1 Distribution des séquences et des ordonnancements sur les nœuds

```
1
2  // λ contient le tour géant (λ_1, ..., λ_N)
3  SPLIT-VRP(λ)
4  {
5    for(i=0;i≤N;i++) node[i] ← ∅;
6    for(i=0;i<N;i++)
7      for(j=i+1;j≤N;j++)
8      {
9        TURN ← (λ_{i+1}, ..., λ_j);
10       if (FEASIBLE? TURN)
11       {
12         PARTIAL-TOUR ← ADD-TURN(node[i], TURN);
13         REPLACE-IF-BETTER(node[j], PARTIAL-TOUR);
14       }
15     }
16
17   return node[N];
18 }
```

$(\lambda_1, \ldots, \lambda_i), i \in [1, k-1]$ soit le meilleur découpage de $(\lambda_1, \ldots, \lambda_i)$ (sinon il y aurait contradiction en le remplaçant puisque le coût est additif). Or, à la fin de l'étape k, le nœud λ_k contient par construction le meilleur découpage possible parmi l'ensemble des meilleurs découpages de $(\lambda_1, \ldots, \lambda_i), i \in [1, k-1]$ suivi du tour $(\lambda_{i+1}, \ldots, \lambda_k)$. Ainsi à la fin de l'étape k le nœud λ_k contient le meilleur découpage de $(\lambda_1, \ldots, \lambda_k)$. Comme l'hypothèse d'induction est vérifiée pour $k = 1$, on obtient le résultat. ∎

Exemples de SPLIT pour le VRP

Soient 5 clients A, B, C, D, E de commandes respectives $4, 13, 8, 5, 10$ et supposons que les véhicules ont une capacité maximale de 22 unités. La matrice des distances est la suivante où O désigne le dépôt :

	O	A	B	C	D	E
O	0	39	43	38	39	43
A	39	0	40	47	36	50
B	43	40	0	46	50	49
C	38	47	46	0	40	39
D	39	36	50	40	0	47
E	43	50	49	39	47	0

Appliquons l'algorithme du SPLIT sur le tour géant (A, B, C, D, E).

(A, B) représente le tour $O \to A \to B \to O$.

A l'étape 1, on distribue les tours $(A), (A, B), (A, B, C), (A, B, C, D), (A, B, C, D, E)$ sur les nœuds A, B, C, D, E à partir du nœud $\varnothing$. Le tour (A, B, C) et les suivants sont impossibles car de

capacité supérieure à 22. On obtient les tournées partielles $[(A), 78]$ et $[(A, B), 122]$.

A l'étape 2, on distribue les tours $(B), (B, C), (B, C, D), (B, C, D, E)$ sur les nœuds B, C, D, E à partir du nœud A. Les tours (B, C, D) et suivants excèdent la capacité et sont donc impossibles. Comme $(A)(B)$ est de coût 164 il ne remplace pas (A, B) de coût 122 dans le nœud B. On obtient la tournée partielle $[(A)(B, C), 205]$ dans le nœud C.

L'étape 3 consiste à distribuer les tours $(C), (C, D), (C, D, E)$ à partir du nœud B. Le tour (C, D, E) demande une capacité de 23 supérieure à 22 et est donc impossible. $[(A, B)(C), 198]$ remplace $[(A)(B, C), 205]$ dans le nœud C. Le nœud D contient maintenant $[(A, B)(C, D), 239]$.

A l'étape 4, on distribue à partir du nœud C les tours $(D), (D, E)$ qui sont tout les deux faisables car de capacité inférieure à 22. $[(A, B)(C)(D), 276]$ ne remplace pas $[(A, B)(C, D), 239]$ dans le nœud D. Le nœud E contient maintenant $[(A, B)(C)(D, E), 327]$.

A la dernière étape, on distribue (E) à partir du nœud D et $[(A, B)(C, D)(E), 325]$ remplace $[(A, B)(C)(D, E), 327]$. Ainsi, le meilleur découpage du tour géant (A, B, C, D, E) est $(A, B)(C, D), (E)$ de coût 325.

	A (4)	B (13)	C (8)	D (5)	E (10)
Étape 1	$[(A), 78]$	$[(A, B), 122]$	$\varnothing$	$\varnothing$	$\varnothing$
Étape 2	$[(A), 78]$	$[(A, B), 122]$	$[(A)(B, C), 205]$	$\varnothing$	$\varnothing$
Étape 3	$[(A), 78]$	$[(A, B), 122]$	$[(A, B)(C), 198]$	$[(A, B)(C, D), 239]$	$\varnothing$
Étape 4	$[(A), 78]$	$[(A, B), 122]$	$[(A, B)(C), 198]$	$[(A, B)(C, D), 239]$	$[(A, B)(C)(D, E), 327]$
Étape 5	$[(A), 78]$	$[(A, B), 122]$	$[(A, B)(C), 198]$	$[(A, B)(C, D), 239]$	$[(A, B)(C, D)(E), 325]$

Tab. 5.1 Étapes du découpage de (A, B, C, D, E)

	C (8)	A (4)	E (10)	D (5)	B (13)
Étape 1	$[(C), 76]$	$[(C, A), 124]$	$[(C, A, E), 178]$	$\varnothing$	$\varnothing$
Étape 2	$[(C), 76]$	$[(C, A), 124]$	$[(C, A, E), 178]$	$[(C)(A, E, D), 251]$	$\varnothing$
Étape 3	$[(C), 76]$	$[(C, A), 124]$	$[(C, A, E), 178]$	$[(C)(A, E, D), 251]$	$\varnothing$
Étape 4	$[(C), 76]$	$[(C, A), 124]$	$[(C, A, E), 178]$	$[(C)(A, E, D), 251]$	$[(C, A, E)(D, B), 310]$
Étape 5	$[(C), 76]$	$[(C, A), 124]$	$[(C, A, E), 178]$	$[(C)(A, E, D), 251]$	$[(C, A, E)(D, B), 310]$

Tab. 5.2 Autre exemple avec le tour géant (C, A, E, D, B), le meilleur découpage est $(C, A, E)(D, B)$ de coût 310.

5.2 Mise en œuvre dans le problème d'ordonnancement périodique

Nous allons maintenant adapter l'algorithme du SPLIT.

Définition 5.5 (Tour géant). *Un* tour géant *est une suite* $(op_{\sigma(1)}, op_{\sigma(2)}, \ldots, op_{\sigma(N)})$ *contenant toutes les tâches du problème.*

Définition 5.6 (Découpages). *Un découpage du tour géant* $(op_{\sigma(1)}, op_{\sigma(2)}, \ldots, op_{\sigma(N)})$ *est une suite* $((op_{\sigma(1)}, \ldots, op_{\sigma(j_1)}), (op_{\sigma(j_1+1)}, \ldots), \ldots) = (\lambda_0^{j_1}, \lambda_{j_1}^{j_2}, \ldots)$*, avec* $\lambda_{j_i}^{j_{i+1}} = (op_{\sigma(j_i+1)}, \ldots, op_{\sigma(j_{i+1})})$*. La concaténation de cette suite redonne le tour géant original.*

On peut voir aussi un découpage comme un tableau de N booléens qui sont vrais pour les valeurs j_i et faux sinon. Ainsi, le vecteur $[t, \ldots, t]$ correspond au découpage $((op_{\sigma(1)}), (op_{\sigma(2)}), \ldots, (op_{\sigma(N)}))$

et le vecteur $[f, \ldots, f, t]$ correspond au découpage $((op_{\sigma(1)}, op_{\sigma(2)}, \ldots, op_{\sigma(N)}))$. Etant donné que $op_{\sigma(N)}$ est toujours la dernière tâche, le dernier booléen est toujours vrai. Le nombre de découpages possibles est donc de 2^{N-1}.

Définition 5.7 (Séquence). *Une séquence de tâches est une suite de tâches dont la première est calée à sa date de disponibilité et les suivantes calées les unes à la suite des autres.*

Une séquence composée d'une tâche d'en-tête op_1 et suivie d'une suite (éventuellement vide) de tâches $op_2, \ldots$ sera notée par la suite $(\underline{op_1}, op_2, \ldots)$.

Dans la démonstration de la condition de faisabilité, nous avons utilisé la séquence $(\underline{op_1}, op_2, \ldots, op_n)$ qui conduit à un ordonnancement de type MacNaughton. Dans le calcul du nombre de machines critiques (voir section 4.8), nous avons utilisé l'ensemble des séquences $\{(\underline{op_1}), (\underline{op_2}), \ldots, (\underline{op_n})\}$. Le critère des intervalles de présence est invariant par regroupement des tâches d'une séquence en une seule tâche pour un utilisateur. On peut associer à chaque découpage un ordonnancement.

Définition 5.8 (Encodage d'un ordonnancement calé à gauche sur des machines homogènes). *On fait correspondre à un découpage $(\ldots, (op_{\sigma(j_1+1)}, \ldots, op_{\sigma(j_{i+1})}), \ldots)$ l'ordonnancement (modulo le placement sur les machines) constitué de l'ensemble des séquences $(\underline{op_{\sigma(j_i+1)}}, \ldots, op_{\sigma(j_{i+1})})$.*

Prenons un exemple, soient les tâches génériques suivantes :

```
Tâches de l'utilisateur A:
  Disponibilité = 0,  Durée = 4, Période = 24
  Disponibilité = 16, Durée = 4, Période = 24
  Disponibilité = 15, Durée = 2, Période = 6

Tâches de l'utilisateur B:
  Disponibilité = 8,  Durée = 5, Période = 12
  Disponibilité = 10, Durée = 3, Période = 8

Période globale = 24
3 machines.
```

Disponibilité	Durée	Utilisateur	Type
0	4	"A"	1
18	3	"B"	5
10	3	"B"	5
2	3	"B"	5
20	5	"B"	4
8	5	"B"	4
21	2	"A"	3
15	2	"A"	3
9	2	"A"	3
3	2	"A"	3
16	4	"A"	2

Tab. 5.3 Tâches de motif

Disponibilité	Durée	Utilisateur	Type	Début	Machine
0	4	"A"	1	0	1
18	3	"B"	5	4	1
10	3	"B"	5	10	1
2	3	"B"	5	2	2
20	5	"B"	4	5	2
8	5	"B"	4	10	2
21	2	"A"	3	21	2
15	2	"A"	3	23	2
9	2	"A"	3	1	3
3	2	"A"	3	3	3
16	4	"A"	2	16	3

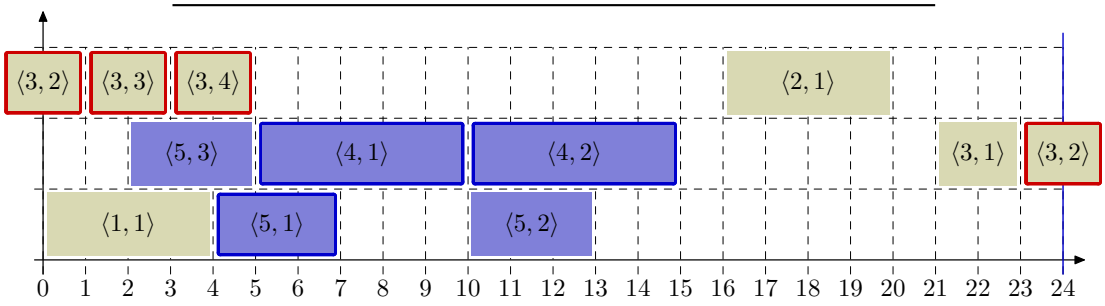


Fig. 5.2.1 Exemple de découpage. Ordonnancement respectant le tour géant $(op_{1,1}, op_{5,1}, op_{5,2}, op_{5,3}, op_{4,1}, op_{4,2}, op_{3,1}, op_{3,2}, op_{3,3}, op_{3,4}, op_{2,1})$. Les tâches de l'utilisateur A sont en rouge et de l'utilisateur B en bleu. Les tâches sans bordures sont calées sur leurs dates de disponibilité. Les séquences sont $(\underline{op_{1,1}}, op_{5,1})(\underline{op_{5,2}})(\underline{op_{5,3}}, op_{4,1}, op_{4,2})(\underline{op_{3,1}}, op_{3,2}, op_{3,3}, op_{3,4})(\underline{op_{2,1}})$ et correspondent au vecteur de booléens $[f, t, \underline{t}, \underline{f}, f, \underline{t}, \underline{f}, \underline{f}, f, t, t]$.

Ils correspondent aux 11 tâches de motif de la table 5.3. Pour ce tour géant il y a 588 découpages faisables sur 3 machines parmi les 1024 découpages possibles. Un exemple d'ordonnancement

périodique qui respecte le tour géant donné est représenté sur la figure 5.2.1.

5.3 Données du problème périodique

Notations

Soit T le PPCM des périodes globales d'arrivée des tâches génériques (on peut aussi prendre pour T un multiple du PPCM). Chaque tâche générique i se rencontre $K_i = T/\alpha_i$ fois dans la période T. Etant donné que l'on travaille sur un motif de période T, chaque tâche est répliquée K_i fois en *tâches de motif* en décalant les dates de disponibilité. La tâche $\{i, j\}$ (de type i et d'index j) désigne ici la j^e apparition de la tâche i répliquée dans le motif. On renumérote les tâches une fois cette réplication effectuée de façon à n'avoir qu'un seul indice ; mais il est possible de faire la correspondance avec la notation $\{type, index\}$. Chaque tâche de motif est donc indexée par un indice unique.

Les données du problème sont donc les suivantes :

Durée de la tâche i : p_i

Disponibilité de la tâche i : r_i (modulo la période T)

Date de début d'exécution de la tâche i : t_i (modulo la période T)

5.4 Extension d'un ordonnancement et dominance globale

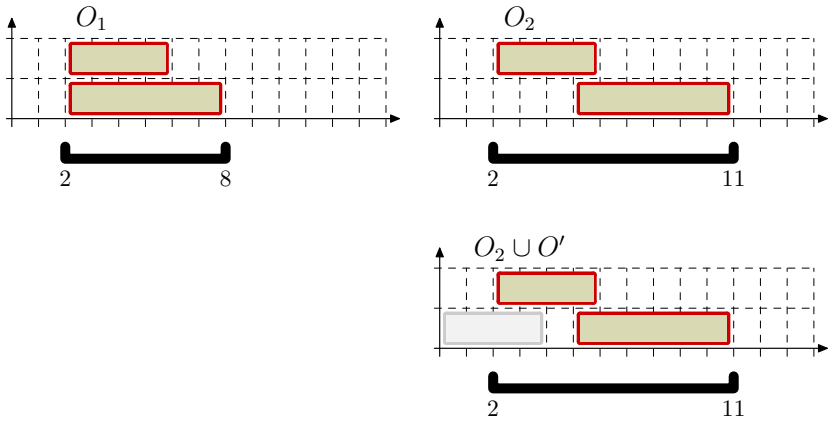


Fig. 5.4.1 O_1 domine O_2 ($LIP(O_1) \subset LIP(O_2)$). Si O' est constitué d'une tâche de durée 4 placée à la date 0 alors $O_1 \cup O'$ est faisable sans que $O_2 \cup O'$ le soit pour $m = 2$ machines.

Définition 5.9 (Dominance locale sur les ordonnancements)**.** *Un ordonnancement O_1 domine* localement *un ordonnancement O_2 (noté $O_1 <^{local} O_2$) si et seulement si pour tout utilisateur U, $LIP_U(O_1) \subset LIP_U(O_2)$ (où LIP_U signifie Liste d'Intervalles de Présence de l'utilisateur U).*

Ainsi si un ordonnancement est localement dominé par un autre, il le sera aussi pour tout

critère croissant pour les listes d'intervalles de présence. L'ordonnancement qui maximise un de ces critère est donc obligatoirement non dominé localement.

De plus comme $LIP_U(O \cup O') = LIP_U(O) \cup LIP_U(O')$, si O_1 domine localement O_2 alors pour tout O' $O_1 \cup O'$ domine localement $O_2 \cup O'$, ie $O_1 \prec^{local} O_2 \Rightarrow \forall O', O_1 \cup O' \prec^{local} O_2 \cup O'$. Nous avons de même l'équivalence puisque O' peut être l'ordonnancement vide.

Toutefois le concept de dominance locale n'est pas suffisant pour garantir l'optimalité en cas d'extensions, lorsqu'il existe des contraintes de faisabilité. Comme nous allons le voir par la suite, il faut introduire une notion de dominance globale.

En effet, du fait de la contrainte sur le nombre de machines, il est possible que $O_2 \cup O'$ soit faisable sans que $O_1 \cup O'$ le soit. Comme il est possible que $O_1 \prec^{local} O_2$ et que O_1 ne puisse se prolonger avec O' alors que O_2 le peut (voir figure 5.4.1). Le fait de garder seulement les solutions non localement dominées par nœuds ne suffit pas (la frontière de Pareto par nœud pour l'ordre d'inclusion des intervalles de présence) et ne donne pas au final la meilleure solution.

De plus autant la propriété de dominance locale s'applique pour les listes d'intervalles de présence, autant elle n'est plus vérifiée pour un autre critère. De fait les critères considérés sont bien *réguliers* dans le sens qu'ils sont croissants par extension ($f(O) \leq f(O \cup O')$). Cependant l'ordre relatif entre ordonnancement n'est pas toujours conservé : on peut avoir $f(O_1) \leq f(O_2)$ et $f(O_1 \cup O') \not\leq f(O_2 \cup O')$ comme le montre la figure 5.4.2. En revanche, si on a la dominance locale alors l'extension conserve l'ordre en raison des propriétés des intervalles de présence. Ainsi, on considére la notion de dominance au sens des intervalles de présence et non au sens du critère particulier f. Le critère f intervient tout de même en filtrant les solutions partielles de moins bonne qualité que la meilleure solution trouvée au cours de la recherche, ceci parce que les critères sont croissants et qu'une solution partielle ne peut que se dégrader par extension. Donc si une solution partielle est de moins bonne qualité qu'une solution elle ne pourra pas être de meilleure qualité après extension.

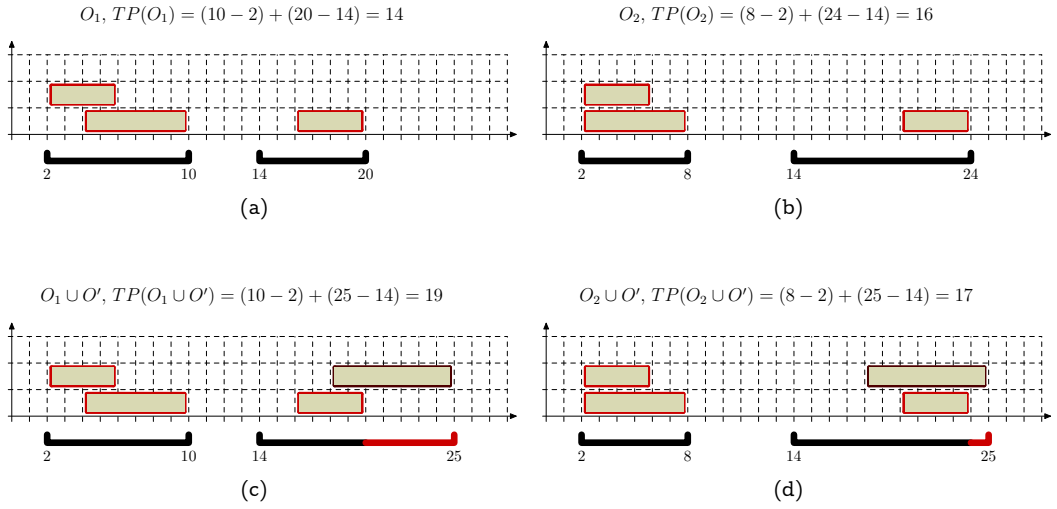


Fig. 5.4.2 Le critère temps de présence TP est croissant si on ajoute des tâches (ie $TP(O) \leq TP(O \cup O')$) mais il ne conserve pas l'ordre entre O_1 et O_2 : $TP(O_1) = 14 < TP(O_2) = 16$. Cependant, après extension avec l'ordonnancement O' composé d'une seule tâche, on obtient que $TP(O_1 \cup O') = 19 > TP(O_2 \cup O') = 17$. Remarquons bien que ni O_1 ni O_2 ne domine localement l'autre.

Prenons un exemple pour bien comprendre l'interêt de la dominance globale, remarquons que ce n'est pas parce que O_1 est de coût inférieur à O_2 (ie $f(O_1) < f(O_2)$) que l'ordre est conservé par extension (ie $\forall O', f(O_1 \cup O') < f(O_2 \cup O')$) [2] ! Prenons l'exemple de la figure 5.4.2 où sont représentées seulement les tâches d'un utilisateur particulier. O_1 et O_2 sont comparables pour la fonction temps de présence parce qu'ils contiennent les mêmes tâches $\{1, 2, 3\}$ (placées differemment). Les tâches d'un utilisateur sont :

$r_1 = 2, p_1 = 8$
$r_2 = 2, p_2 = 6$
$r_3 = 14, p_3 = 4$
$r_4 = 18, p_4 = 7$

Les temps de présence de l'utilisateur sont $TP(O_1) = 14 < TP(O_2) = 16$. O' consiste en une tâche $\{4\}$ placée à $t = 18$. Après l'ajout de O', les temps de présence sont $TP(O_1 \cup O') = 17 > TP(O_2 \cup O') = 16$. Ainsi, le critère temps de présence est croissant pour l'ordre d'inclusion des intervalles (ie $TP(O) \leq TP(O \cup O')$) mais ne respecte pas l'ordre entre les intervalles après union (ie dans l'exemple ici $TP(O_1) = 14 < TP(O_2) = 16$ et $TP(O_1 \cup O') = 19 > TP(O_2 \cup O') = 17$). Remarquons qu'ici O_1 ne domine pas localement O_2 (au sens des intervalles de présence).

Ce serait vrai si on impose que O_1 et O_2 soient inclus dans un intervalle $I = [0, t]$ et que O' ne contient que des tâches dont l'intervalle de présence est en dehors de l'intervalle I. Mais cela est rendu doublement difficile par le caractère périodique qui peut recouper en début de I comme à la fin. De plus, on ne peut pas en général partitionner ainsi les tâches.

Pour corriger ces défauts nous allons introduire la notion de dominance globale. Nous allons mentionner des conditions pour que la faisabilité et l'ordre entre ordonnancements soient conservés après extension malgré les contraintes et ainsi obtenir une condition de dominance sur les ordonnancements partiels.

Remarquons tout d'abord que si O_1 et O_2 ont les mêmes séquences à une permutation des tâches dans les séquences près sauf la première tâche alors O_1 et O_2 utilisent le même nombre de machines au cours du temps. Par conséquent $O_1 \cup O'$ est faisable si et seulement si $O_2 \cup O'$ est faisable. Dans ce cas si l'extension d'un des ordonnancement est faisable l'autre l'est aussi. Si l'un domine l'autre, il le domine pour toutes les extensions possibles et ne peut être infaisable sans que l'autre le soit. La propriété de dominance globale est similaire et généralise cette idée :

Définition 5.10 (Dominance globale sur les ordonnancements)**.** *Un ordonnancement O_1 domine* globalement *un ordonnancement O_2 (noté $O_1 \prec O_2$) si et seulement si quelque soit O' avec $O_2 \cup O'$ faisable alors $O_1 \cup O'$ est faisable et on a $O_1 \cup O' \prec^{local} O_2 \cup O'$.*

Remarquons que si O' est l'ordonnancement vide, on doit avoir $O_1 \prec^{local} O_2$ soit $LIP_U(O_1) \subset LIP_U(O_2)$.

La dominance locale ne garantit pas d'avoir une extension réalisable pour O_2 sans l'être pour O_1. Ainsi la définition requiert que si O_1 domine globalement un ordonnancement O_2 alors pour toute extension réalisable de O_2 avec O', $O_1 \cup O'$ est réalisable et domine $O_2 \cup O'$.

Cette propriété garantit que les ordonnancements partiels globalement dominés peuvent être éliminés au cours de la recherche.

Notons que dans la définition rien n'impose le fait que les deux ordonnancements aient le même ensemble de tâches, d'ailleurs :

2. Ceci rend difficile l'algorithme du SPLIT.

Théorème 5.2 (Dominance locale et ordonnancements sur un sous-ensemble de tâches)**.** *Soit deux ordonnancements O_1 et O_2 avec*

- *l'ensemble des tâches de O_2 est inclus dans l'ensemble des tâches de O_1*
- *et O_1 domine globalement O_2.*

Alors O_1 domine globalement tout ordonnancement faisable $O_2 \cup O$ où O est constitué des tâches de O_1 qui ne sont pas dans O_2.

Démonstration. Soit E l'ensemble des tâches de O_1 qui ne sont pas dans O_2 et soit O un ordonnancement des tâches de E tel que $O_2 \cup O$ est réalisable. Soit O' un ordonancement tel que $O_2 \cup O \cup O'$ est réalisable, alors comme O_1 domine globalement O_2 et que $O_2 \cup O \cup O'$ est réalisable, $O_1 \cup O \cup O'$ est faisable et domine localement $O_2 \cup O \cup O'$. Or, les tâches de l'ensemble E sont en double dans l'ordonnancement $O_1 \cup O \cup O'$. En supprimant O, on obtient un ordonnancement $O_1 \cup O'$ faisable qui domine localement $O_1 \cup O \cup O'$, lequel domine localement $O_2 \cup O \cup O'$. Ainsi, pour tout ordonnancement O des tâches de E avec $O_2 \cup O$ faisable, O_1 domine globalement $O_2 \cup O$. ∎

Il nous faut donc exprimer le fait qu'un ordonnancement domine un autre quelle que soit la façon d'étendre ces ordonnancements. Pour cela introduisons la notion de l'utilisation de machines au cours du temps :

Définition 5.11 (Nombre de machines au cours du temps)**.** *Soit O un ordonnancement. Le nombre de machines qu'utilise O au temps t est noté $\mu_O(t)$. Le nombre de machines requises pour O (noté $m(O)$) est le nombre maximal de machines utilisées par O, soit :* $m(O) = \max_t \mu_O(t)$.

Propriétés :

- $\mu_{O_1 \cup O_2}(t) = \mu_{O_1}(t) + \mu_{O_2}(t)$
- $m(O_1 \cup O_2) \leq m(O_1) + m(O_2)$
- $\sum_t \mu_O(t) = \sum_{i \in O} p_i$

Ainsi, avec M machines si $m(O_1) + m(O_2) \leq M$, on a $m(O_1 \cup O_2) \leq M$ et $O_1 \cup O_2$ est faisable. C'est une condition suffisante mais pas nécessaire, il suffit pour s'en convaincre que O_1 utilise toutes les machines sur l'intervalle I_1 et aucune machine sur un intervalle I_2 disjoint de I_1 et inversement pour O_2.

Cela signifie que si on étend un ensemble d'ordonnancements avec O' qui utilise au plus m' machines, tous les ordonnancements utilisant au plus moins de $M - m'$ machines pourront être étendus.

Théorème 5.3 (Conditions pour la dominance globale)**.** *Si O_1 et O_2 utilisent le même ensemble de tâches, avec O_1 qui utilise moins de machines au cours du temps que O_2 (ie $\forall t, \mu_{O_1}(t) \leq \mu_{O_2}(t)$) et si O_2 est dominé localement par O_1, alors O_1 domine globalement O_2.*

Démonstration. Comme O_1 domine localement O_2 et que le fait d'utiliser moins de machines au cours du temps impose que tout ordonnancement faisable pour O_2 l'est pour O_1 ; on a la dominance globale. ∎

Lemme 5.4 (Maximums locaux de μ_O)**.** *Pour les ordonnancements dont les tâches sont calées à gauche les maximums locaux se trouvent parmi les dates de disponibilité des tâches.*

Démonstration. μ_O est décroissante entre 2 dates de disponibilité pour les ordonnancements dont les tâches sont calées à gauche et augmente éventuellement de 1 à une date de disponibilité. Les maximums locaux sont donc parmi les dates de disponibilité de μ_O. ∎

Théorème 5.5 (Propriété du nombre de machines au cours du temps). *Soient O_1 et O_2 deux ordonnancements de mêmes tâches et calés à gauche (chaque tâche commence soit à sa date de disponibilité soit à la fin d'une autre tâche). Soit O' un ordonnancement. Soient $\{r_1, \ldots, r_n\}$ l'ensemble contenant les dates de disponibilité des tâches de O' ainsi que les maximaux locaux de μ_{O_1} et μ_{O_2} (qui sont parmi les dates de disponibilité des tâches de O_1 et O_2). Alors :*

$$\forall t \in \{r_1, \ldots, r_n\}, \mu_{O_1}(t) \leq \mu_{O_2}(t) \quad \Rightarrow \tag{5.4.1}$$

$$(m(O_2 \cup O') \leq M \quad \Rightarrow \quad m(O_1 \cup O') \leq M) \tag{5.4.2}$$

ie toute extension faisable pour O_2 l'est alors pour O_1.

Démonstration. Puisqu'on rajoute seulement les séquences aux dates de disponibilité des tâches de O', si la propriété 5.4.1 est vérifiée alors une extension faisable pour O_2 l'est pour O_1. En effet, pour tout t, on a $\mu_{O_2 \cup O'}(t) = \mu_{O_2}(t) + \mu_{O'}(t) \leq M$ car $O_2 \cup O'$ est faisable par hypothèses. Pour $t \in \{r_1, \ldots, r_n\}$, $\mu_{O_1 \cup O'}(t) = \mu_{O_1}(t) + \mu_{O'}(t) \leq \mu_{O_2}(t) + \mu_{O'}(t) \leq M$ pour chaque date dans $\{r_1, \ldots, r_n\}$.

Les ordonnancements $O_1 \cup O'$ et $O_2 \cup O'$ sont calés à gauche, par conséquent leurs maximums locaux se trouvent dans $\{r_1, \ldots, r_n\}$. On a donc $\exists r \in \{r_1, \ldots, r_n\}, m(O_1 \cup O') = \max_t \mu_{O_1 \cup O'}(t) = \mu_{O_1 \cup O'}(r) = \mu_{O_1}(r) + \mu_{O'}(r) \leq \mu_{O_2}(r) + \mu_{O'}(r)$, d'où si $O_2 \cup O'$ est faisable, on a $\mu_{O_2}(r) + \mu_{O'}(r) \leq M$ et $O_1 \cup O'$ est faisable aussi.

■

Cela implique qu'il suffit uniquement de vérifier le nombre de machines que nécessite un ordonnancement sur les dates de disponibilité. Par exemple, dans le cas particulier où toutes les tâches arrivent en début de période il suffit de vérifier uniquement à la date 0. On en déduit que :

Théorème 5.6 (Conditions suffisantes pour la dominance globale). *Si O_1 domine localement O_2 avec $\forall t \in \{r_1, \ldots, r_n\}, \mu_{O_1}(t) \leq \mu_{O_2}(t)$ alors O_1 domine globalement O_2.*

Cette propriété est d'autant plus efficace qu'il y a peu de dates de disponibilité différentes.

Le résultat pourrait être affiné si on connaissait une borne sur le nombre de machines que tout O' pourra utiliser à une date t donnée. Dans ce cas, certaines dates de disponibilité auraient toujours moins de M machines et pourraient ne pas être prises en compte, i.e. : si $\forall O', \mu_{O'}(t) \leq M'(t)$ alors $\mu_{O_1}(t) \leq M - M'(t) \Rightarrow \mu_{O_1 \cup O'}(t) \leq M$ et $O_1 \cup O'$ est toujours faisable à cet instant t. Seulement il faut être capable de borner le nombre de machines que peut prendre O' à un instant t. Une borne triviale est le nombre N de tâches restantes à placer.

Prenons un cas simple où toutes les tâches ont une date de disponibilité égale à 0. A l'instant t le nombre maximum de machines que peut prendre un ordonnancement est au plus égal à $\min(\sum p_i/t, N)$, car si une machine est occupée à l'instant t elle est occupée dans l'intervalle $[0, t]$, et si il y a M' machines occupées à l'instant t la quantité de calcul doit être d'au moins $M't$. Comment généraliser ce résultat avec plusieurs dates de disponibilité ? Soit t' la date de disponibilité la plus proche de t et inférieure à t, alors si M' machines sont occupées à t' on a une quantité de calcul d'au moins $M'(t - t')$. Si l'ordonnancement n'est pas périodique on peut donc se limiter au nombre de tâches qui ont une date de disponibilité entre t' et t. Notons que la borne est mauvaise si $Q \approx MT$

Il n'est pas suffisant de regarder seulement aux dates de disponibilité des tâches de O' : Supposons que $\mu_{O_1}(r_i) \leq \mu_{O_2}(r_i)$ avec $\mu_{O_1}(r_i + 1) = M$ et $\mu_{O_2}(r_i + 1) < M$, alors ajouter la tâche i en la calant à sa date de disponibilité r_i est possible pour O_2 mais pas pour O_1.

Par contre, remarquons que si une date de disponibilité n'est pas utilisée pour caler une tâche alors cette date n'est pas maximum local de μ pour cet ordonnancement. Si elle est utilisée, il n'y a pas forcément un maximum local car il suffit qu'une séquence se termine a cette date.

Proposition :

Comme on souhaite comparer lorsque toutes les tâches sont présentes et qu'on connait des intervalles où l'utilisateur est présent, on compare O_1 et O_2 en rajoutant l'union des intervalles de présence avec l'intervalle de présence minimale par utilisateur.

Il faut prendre le maximum pour le critère f à la fin, une fois le front de Pareto calculé.

5.4.1 Pré-traitement du problème

Le problème est infaisable si le nombre de machines est insuffisant : $M < \frac{\sum p_i}{T}$. Si le nombre de machines est suffisant, on calcule ensuite les intervalles de présence minimale grâce aux bornes obtenues avec $M = 1$ et $M = \infty$. Si tous les utilisateurs sont permanents, tous les ordonnancements sont indiscernables pour le critère.

Si ce n'est pas le cas, c'est qu'il existe des utilisateurs potentiellement non permanents. On essaye alors d'allouer un ensemble de machines aux utilisateurs permanents et de caler les tâches des utilisateurs non permanents sur leur date de disponibilité. Si cela est possible, on obtient un ordonnancement optimal.

Si cela n'est pas possible, on poursuit alors sur la méta-heuristique du GRASP-ELS laquelle utilise l'heuristique du SPLIT.

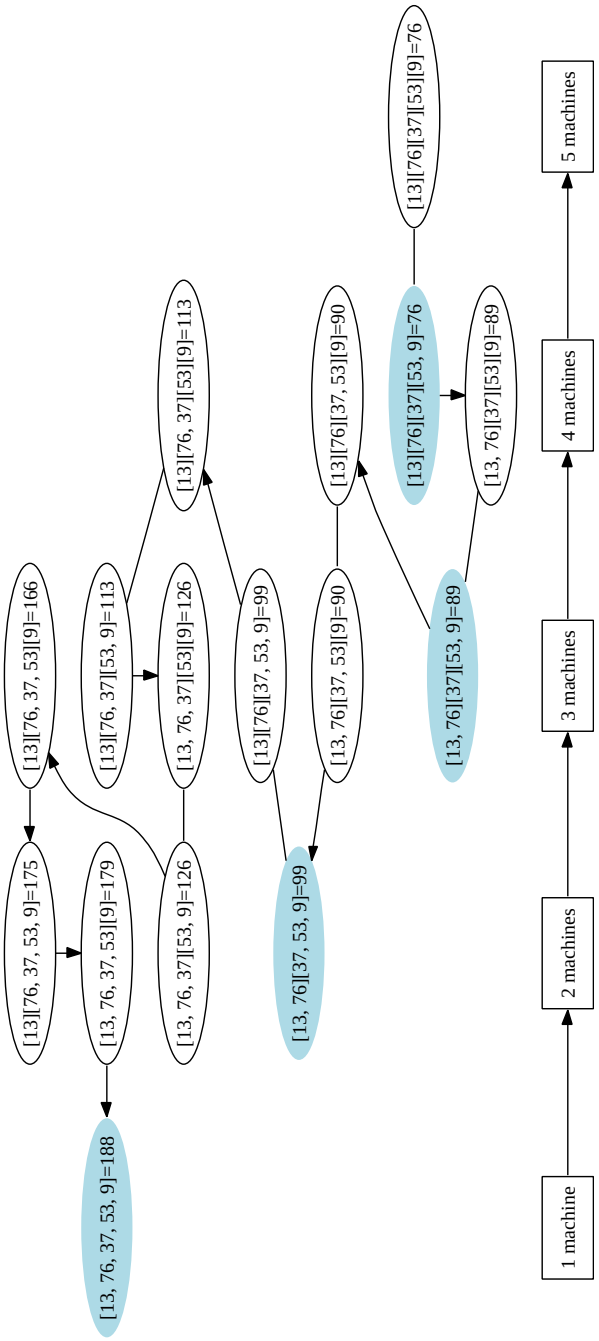


Fig. 5.4.3 Exemple pour le Makespan avec 5 tâches disponibles à la date 0 dont l'ordre de placement est imposé. Le nombre de machines utilisées à la date 0 est indiqué en dessous. Une flèche indique l'ordre selon le critère du Makespan. Les ordonnancements notés avec des cercles bleus sont les ordonnancements globalement dominants pour le nombre de machines considéré.

5.5 Meta-Heuristique employée : "Greedy Randomized Adaptive Search Procedures"

"Greedy Randomized Adaptive Search Procedures" (GRASP) [Hart et Shogan, 1987; Feo et Bard, 1989; Feo et Resende, 1995; Resende, 2000; Festa et Resende, 2002] est une procédure de recherche globale itérative dans laquelle une méthode de recherche locale est appliquée successivement à n solutions fabriquées à partir d'une méthode de résolution stochastique. n points de départ sont générés aléatoirement dans un espace de configurations. Pour chacun de ces points, on applique une méthode de résolution stochastique suivie d'une recherche locale dans laquelle le voisinage d'une solution est exploré. La meilleure solution est selectionnée et la procédure du GRASP peut se poursuivre avec cette nouvelle solution (Voir le pseudocode du listing 5.3).

5.5.1 Références

La méthode GRASP a été utilisée pour le problème de routage de véhicules et de location de dépôts [Duhamel et al., 2010]. [Feo et al., 1991] a recourt au GRASP pour un problème difficile d'ordonnancement sur une machine avec pénalités d'avance et temps de séjour (Flow time). Ils montrent que les solutions trouvées sont proches des optimums connus pour des problèmes de taille moyenne. Dans [Laguna et Velarde, 1991], GRASP a été utilisé pour le problème d'ordonnancement "juste à temps" sur machines parallèles. [Cravo et al., 2008] utilisent le GRASP pour le placement de labels dans la construction de cartes.

5.5.2 Espace des configurations

L'espace des configurations de notre problème est l'ensemble des listes sur les tâches $\{op_1, op_2, \ldots, op_N\}$. Une liste est caractérisée par une permutation σ ; la liste correspondante $(op_{\sigma(1)}, op_{\sigma(2)}, \ldots, op_{\sigma(N)})$ est appelée un *tour géant* en réference aux problèmes de tournées de véhicules [Funke et al., 2005; Gromicho et al., 2009; Duhamel et al., 2010].

5.5.3 "Evolutionary Local-Search" ou ELS

Le principe de ELS se base sur l'idée des "avalanches" : de grandes fluctuations ajustent les paramètres de la recherche afin d'explorer efficacement l'espace des configurations, en essayant de dénombrer, le plus exhaustivement possible dans les limites de la recherche de solutions, les configurations intéressantes. Ces grandes fluctuations permettent à l'algorithme de s'évader de zones d'éventuels minimums locaux, tandis que le processus de sélection associé entraîne l'exploration vers des solutions localement optimales dans un voisinage défini.

Un tour géant est tiré aléatoirement au début de chaque tour du GRASP ; la procédure ELS est appelée avec ce tour géant et commence par diversifer la recherche en générant un voisinage de ce tour géant. Pour chacun de ces tours nous appliquons la méthode du SPLIT. Celle-ci fournit un découpage et l'ordonnancement correspondant, chacun de ces SPLIT peut être exécuté en parallèle.

Pour chacun de ces ordonnancements, plusieurs améliorations locales sont appliquées. A la fin de la procédure ELS, nous conservons le meilleur ordonnancement, une procédure recrée un tour géant correspondant à la meilleure solution trouvée et on poursuit à nouveau la boucle jusqu'à un

Listing 5.2 Algorithme ELS

```
1  ELS(tour-geant, meilleur-ordonnancement, ELS-LIMITE, R_tour)
2  {
3    for(index=1; index≤ELS-LIMITE; index++)
4    {
5      voisinage-tour-geant ← generer-voisinage-tour-geant(tour-geant, R_tour);
6      ensemble-meilleurs-decoupages ← pour-chaque-tour-SPLIT(voisinage-tour-geant,
           meilleur-ordonnancement);
7      ameliorations-locales ←
           pour-chaque-ordonnancement-recherche-locale(ensemble-meilleurs-decoupages);
8
9      meilleur-ordonnancement ← selectionner-meilleur(ameliorations-locales);
10     tour-geant ← ordonnancement-calculer-tour(meilleur-ordonnancement);
11   }
12
13   return meilleur-ordonnancement;
14 }
```

Listing 5.3 Algorithme GRASP

```
1  GRASP(MAX-ELS, nbmachines, meilleur-ordonnancement)
2  {
3    for(index=1; index≤MAX-ELS; index++)
4    {
5      tour-geant ← creer-tour-geant-aleatoire();
6      ordonnancement-initial ← première-estimation(tour-geant, nbmachines);
7      nouvel-ordonnancement ← ELS(tour-geant, ordonnancement-initial, ELS-LIMITE, R_tour);
8      if (meilleur? nouvel-ordonnancement meilleur-ordonnancement)
9      {
10       meilleur-ordonnancement ← nouvel-ordonnancement;
11     }
12   }
13
14   return meilleur-ordonnancement;
15 }
```

nombre d'itérations fixées. Le meilleur ordonnancement trouvé est retourné.

Voisinages du tour géant

Un tour géant est caractérisée pas une permutation σ. Le tour correspondant est $(op_{\sigma(1)}, op_{\sigma(2)}, \ldots, op_{\sigma(N)})$. Le voisinage considéré est l'ensemble des permutations obtenues à partir de σ en appliquant au plus R_{tour} transpositions.

5.5.3.a Recherches locales

Méthode de découpage/recollage

L'idée de cette recherche locale est de tenter de supprimer un point parmi les intervalles de présence d'un utilisateur. Pour ce faire pour chaque utilisateur non permanent un temps t est tiré aléatoirement en dehors des intervalles de présence minimale (car quelque soit l'ordonnancement on sait que l'utilisateur y sera présent). Chaque séquence va être découpée autour de ce point t.

Nous recherchons les tâches de cet utilisateur qui sont présentes à cet instant t. Nous découpons alors chaque séquence autour de ces tâches. Par exemple : soit une séquence $(\underline{op_1}, op_2, \ldots, op_a, \ldots)$ avec op_a présente à l'instant t. Nous obtenons alors les séquences $(\underline{op_1}, op_2, \ldots, op_{a-1})$ et $(\underline{op_a}, \ldots)$. Le découpage se poursuit ainsi de suite.

En général, ce nouvel ensemble de séquences n'est pas faisable avec M machines. On passe alors à la phase de recollage, les séquences sont mélangées et concaténées deux à deux aléatoirement. Au bout d'une certaine limite d'itérations qu'on peut spécifier, la recherche se termine et la meilleure solution est retournée.

La méthode est efficace pour les tâches de courte durée qui ont un temps de présence important dans l'ordonnancement : ces tâches peuvent alors être regroupées dans d'autres intervalles de présence.

Permutation des tâches entre séquences

La seconde méthode appliquée consiste à permuter aléatoirement des tâches entre séquences ; éventuellement la même. Au bout d'une certaine limite d'itérations, la recherche se termine et on retourne la meilleure solution trouvée.

Permutation des tâches de chaque séquence

La troisième méthode s'appuie sur la remarque suivante : un découpage faisable reste faisable si les tâches de chaque séquence sont permutéee en conservant la première tâche de la séquence, qui est calée sur sa date de disponibilité. La recherche locale consiste alors à permuter les tâches à l'intérieur de chaque séquence sans déplacer la première tâche. Cela permet d'économiser un test de faisabilité du découpage obtenu. La meilleure solution est retournée.

5.6 SPLIT appliqué au problème d'ordonnancement périodique

5.6.0.b Méthode de résolution stochastique

Dans notre cas la méthode de résolution stochastique se compose d'une méthode de résolution exacte sous contraintes de tour géant (SPLIT). La méthode de résolution exacte est exécutée de façon limitée afin de borner les temps de calcul LIMITED-SPLIT. Les solutions ainsi trouvées ne sont pas forcément optimales. Pour pallier à ce fait, une recherche locale amène la solution vers un optimum local. Celui-ci conduit à un nouveau point de l'espace des configurations à partir duquel la méthode de résolution stochastique est de nouveau appliquée.

Au début de l'heuristique, plusieurs tours géant sont tirés au hasard. Pour chacun de ces tours, on génére d'autres tours voisins afin de diversifier les solutions. Pour chacun de ces tours géants, une recherche locale couplée à un algorithme de résolution est appliquée.

La procédure SPLIT calcule à partir d'un tour géant le découpage correspondant à un ordonnancement optimal sous la contrainte de ce tour géant.

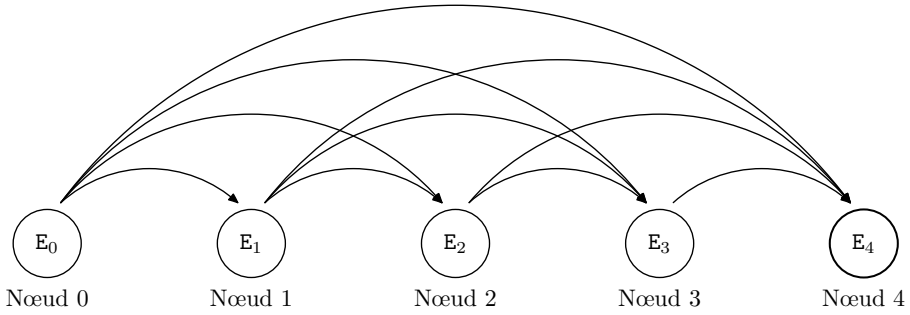


Fig. 5.6.1 Étapes complètes du SPLIT pour 4 tâches.

LIMITED-SPLIT est une procédure identique en fonctionnement à SPLIT excepté le fait qu'elle applique une limite à la distribution des nœuds. Pour chaque nœud seuls les K premiers éléments sont distribués sur les nœuds suivants, K étant un paramètre de la procédure.

Un ensemble de recherches locales sont réalisées sur l'ordonnancement obtenu par la procédure LIMITED-SPLIT et la meilleure solution est conservée.

On extrait un tour géant que respecte cette solution et la méthode est réappliquée sur ce nouveau tour un certain nombre de fois.

5.6.1 Adaptation du SPLIT dans le cas de tâches périodiques

L'idée est de considérer les ordonnancements qui respectent une séquence géante imposée. Le tour géant est donné en paramètre de la procédure SPLIT.

Un ordonnancement respecte une séquence géante $(op_1, \ldots, op_N)$ si pour toutes les tâches soit la tâche est la dernière d'une séquence, soit la suivante est celle spécifiée par la séquence géante.

Chaque nœud i représente l'ensemble des sous-ordonnancements qui respectent la sous-séquence $(op_1, \ldots, op_i)$. Un élement de ce nœud est un label composé d'un ordonnancement qui respectent la sous-séquence $(op_1, \ldots, op_i)$, ainsi qu'un vecteur indexé par les utilisateurs. Ce vecteur contient la liste des intervalles de présence de cet utilisateur.

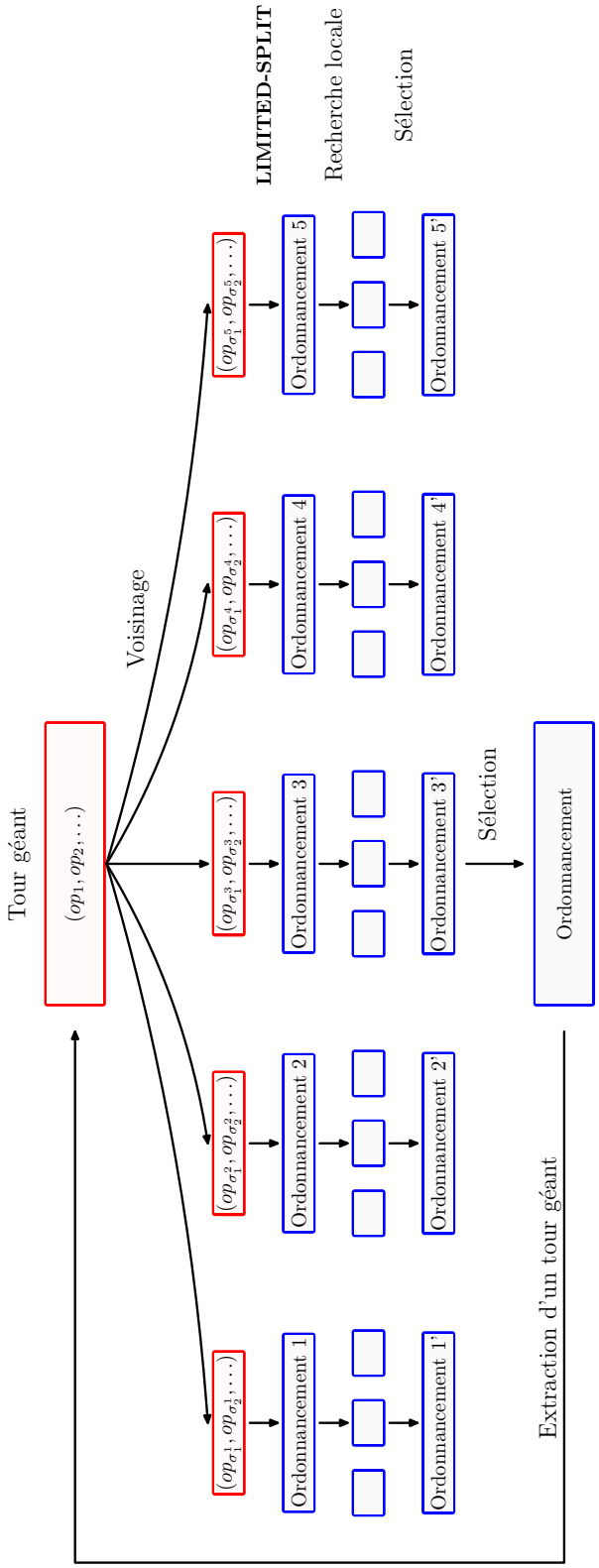


Fig. 5.6.2 Schéma de la méthode du GRASP-ELS appliquée au SPLIT.

Dans un premier temps l'algorithme se contente d'énumérer toutes les possibilités et retourne au nœud N, l'ensemble des ordonnancements qui respectent la séquence géante.

Une heuristique supplémentaire consiste à garder dans les nœuds uniquement les ordonnancements non dominés pour l'ordre d'inclusion des listes d'intervalles de présence. Comme remarqué plus haut, garder seulement la valeur de la fonction objectif ne suffit pas puisqu'elle n'est pas croissante avec l'union d'ordonnancements. D'autre part, se pose le problème des ordonnancements non dominés qui deviennent irréalisables une fois étendu avec une séquence. Cela a pour conséquence que le résultat final sur le dernier nœud n'est pas optimal.

La procédure SPLIT fournit toujours une solution

En effet, dans le cas périodique la séquence $(\underline{op_1}, op_2, \ldots, op_N)$ est faisable et se trouve forcément dans le dernier nœud, lequel contient au final au moins un résultat.

5.6.2 Front de Pareto

L'ordre de comparaison est l'ordre coordonnées par coordonnées $\prec_c$. On construit le front de Pareto par ajout successif d'un élément e à un front de Pareto. En enumérant les éléments du front de Pareto initial on les classe en 3 sous-ensembles, ceux dominés par e, ceux qui dominent e et ceux qui sont incomparables avec e. Si on rencontre un e' qui domine e on arrête, et on retranche les éléments dominés.

5.7 Algorithme du SPLIT en ordonnancement

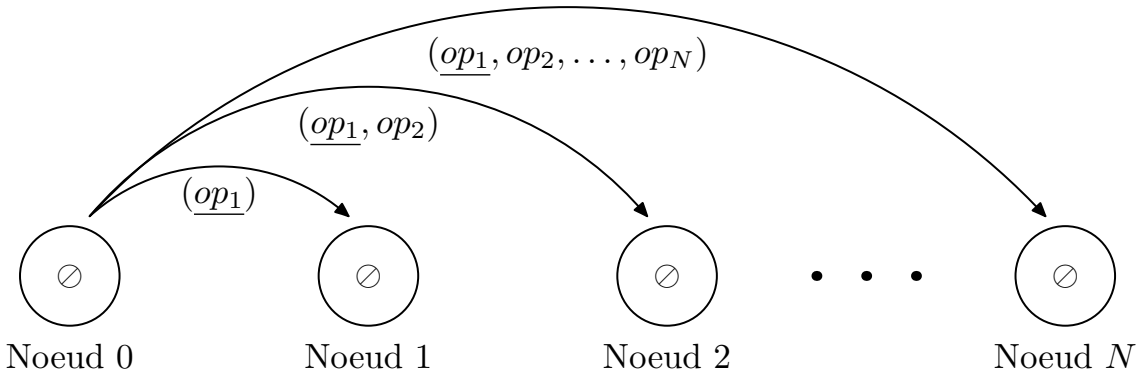


Fig. 5.7.1 Étape 0

Enumération complète

A des fins de debogage, l'énumération compléte est possible en désactivant le filtrage lors de l'ajout d'un ordonnancement dans un label.

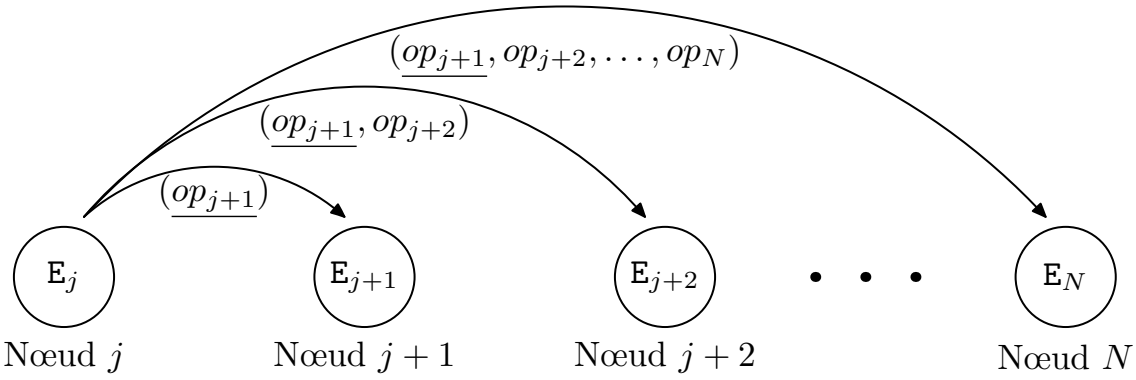


Fig. 5.7.2 Étape j

5.7.1 Ordonnancement initial employé pour le filtrage des solutions partielles

Théorème 5.7 (Filtrage des ordonnancements partiels)**.** *Soit un ordonnancement partiel O et un ordonnancement O^* contenant toutes les tâches. Si $f(O) \geq f(O^*)$, alors on peut éliminer O de la recherche.*

Démonstration. Comme f est croissante, on a pour toute extension O', $f(O \cup O') \geq f(O) \geq f(O^*)$ et $O \cup O'$ ne peut pas être meilleur que O^*. ∎

Nous utilisons cette propriété pour filtrer dans chaque nœud les solutions partielles dominées par la meilleure solution trouvée jusqu'à présent. D'où l'interêt d'avoir dès le départ de l'heuristique une solution de bonne qualité. Nous proposons une solution construite de la façon suivante : nous relaxons le problème en considérant qu'il n'y a qu'un seul utilisateur et que l'objectif est de minimiser le Makespan, en respectant la contrainte de tour géant, ceci afin d'avoir une solution de départ relativement "compacte".

On définit le Makespan d'un ordonnancement périodique comme la date de fin de la dernière tâche parmi les tâches reçues dans une période. L'idée est de chercher un ordonnancement de Makespan inférieur à C fixé. Une procédure permet de savoir si c'est possible ou non, puis une recherche dichotomique sur C est effectuée afin de trouver le C le plus petit possible.

La procédure qui permet de savoir s'il existe un découpage de Makespan inférieur à C se base sur le test rapide suivant : si l'ajout d'une tâche o à la suite de la séquence courante S vérifie la contrainte de Makespan inférieur à C, on l'ajoute et on poursuit avec la tâche suivante. On continue ainsi et si l'ordonnancement final respecte C et utilise moins de M machines on le garde, sinon on sort de la procédure avec le booléen *false*. Il est possible que la procédure ne trouve pas un découpage de Makespan inférieur à C lorsqu'il en existe un faisable à cause des dates de disponibilité, mais l'important ici est de trouver rapidement une solution initiale de qualité raisonnable.

D'autre part, en plus de cette solution on génére aléatoirement un certain nombre de découpages, dont le nombre est spécifié en paramètre de l'heuristique. On ne garde que les découpages qui utilisent moins de M machines et on conserve pour la solution de filtrage le meilleur ordonnancement trouvé.

5.7.2 Dominances utilisées dans l'algorithme du SPLIT

Utilisateurs permanents

Selon le théorème 4.20, on sait qu'il suffit de ne caler que les tâches des utilisateurs non permanents. Par conséquent, le nœud i est inutile lorsque la tâche $op_{\sigma(i+1)}$ appartient à un utilisateur permanent. On veillera donc à ce que le tour géant commence par une tâche d'un utilisateur non permanent. De plus, l'arc du nœud a au nœud b contiendra la séquence $(\underline{op_{\sigma(a+1)}}, \ldots, op_{\sigma(b)}, op_{\sigma(b+1)}, \ldots, op_{\sigma(c)})$ au lieu de $(\underline{op_{\sigma(a+1)}}, \ldots, op_{\sigma(b)})$, si les tâches $op_{\sigma(b+1)}, \ldots, op_{\sigma(c)}$ appartiennent à des utilisateurs permanents et pas $op_{\sigma(c+1)}$.

Elagage grâce au dernier nœud

A chaque itération, la meilleure solution pour le critère retenu est selectionnée dans le dernier nœud. Ainsi, le dernier nœud ne contient qu'un découpage au lieu d'avoir un ensemble de solutions. En effet, cela est suffisant puisqu'on recherche le meilleur découpage selon le critère considéré.

Cette meilleure solution est utilisée dans la prochaine itération afin d'élaguer toutes les solutions partielles dans les nœuds qui sont de moins bonne qualité selon le critère.

Avant la première itération de SPLIT plusieurs ordonnancements aléatoires sont calculés et le meilleur est sélectionné. Cet ordonnancement est inséré dans le dernier nœud afin de filtrer les solutions partielles de moins bonnes. L'élagage se poursuit ainsi au cours de la première itération de boucle et dans les suivantes avec la meilleure solution trouvée à ce point.

Dominances sur les arcs considérés comme des ordonnancements

Remarquons que les arcs (qui sont des séquences à rajouter) peuvent être considéré comme des ordonnancements avec une seule séquence. Ainsi, la meilleure solution trouvée à chaque itération sert aussi à élaguer les arcs dont l'ordonnancement composé de la séquence correspondante est de moins bonne qualité pour le critère considéré.

D'autre part soient deux arcs s_1 et s_2 dont s_1 est une sous-séquence de s_2, si l'ordonnancement constitué de la seule séquence s_2 domine globalement l'ordonnancement constitué du seul arc s_1, alors en vertu du théorème 5.2 on sait que pour tout ordonnancement contenant s_1 il existe un ordonnancement contenant s_2 qui le domine. En effet, il suffit de prendre pour O l'ordonnancement constitué du placement des tâches de s_2 n'étant pas dans s_1, alors l'ordonnancement constitué de la seule séquence s_2 domine globalement O étendu avec la séquence s_1.

Ainsi pendant la phase de distribution des arcs, si deux arcs successifs sont dans ce cas de figure, on ne distribue que l'arc non dominé.

5.7.3 Complexité du SPLIT

Calculons la complexité de l'algorithme SPLIT.

Soit l'étape $i \to j$, Il y a $O(n^2)$ étapes pour n tâches. Le nombre maximal d'élements du nœud i à l'étape j est noté A_i^j. On a $A_0^0 = 1$ et $A_i^0 = 0$ pour $i \geq 1$.

L'ajout des éléments du nœud i augmentés de la séquence $\sigma = (\underline{op_{i+1}}, \ldots, op_j)$ au nœud j donne au pire (si aucun n'est dominé) : $A_k^i = A_k^{i-1} + A_{i-1}^{i-1}$

Ainsi $A_i^i = A_i^{i-1} + A_{i-1}^{i-1} = A_i^0 + \sum_{j=0}^{i-1} A_j^j$

Or $A_i^0 = 0$ pour $i \geq 1$ et $A_0^0 = 1$.
donc $A_i^i = \sum_{j=0}^{i-1} A_j^j$ et pour $i \geq 1$ on a $A_i^i = 2^{i-1}$ et $A_0^0 = 1$.
A l'étape $i \in 0, \ldots, n-1$ on a $A_i^i(n-i)$ tâches au pire.
Soit un total de $Total = \sum_{i=0}^{n-1}(n-i)A_i^i = n + \sum_{i=1}^{n-1}(n-i)2^{i-1}$, en inversant lignes et colonnes, $Total = n + \sum_{j=1}^{n-1}\sum_{i=1}^{n-j} 2^{i-1} = 2^{n+1} - 2$ tâches.
La complexité de l'algorithme au pire est donc de $2^{n+1} - 2 = O(2^n)$ tâches.
Etant donné que le nombre d'élements d'un nœud peut être exponentiel, il est conseillé de limiter la taille des labels. Par exemple, on pourra conserver au plus K solutions dans chaque nœud.

Complexité de la procédure LIMITED-SPLIT

La procédure LIMITED-SPLIT limite chaque nœud à K éléments. Parmi ces K éléments, on garde uniquement les éléments non dominés (la frontière de Pareto), ce qui se fait pour chaque nœud en $O(K^2)$ tâches de comparaison, soit $O(NK^2)$ tâches. De plus, on distribue les éléments restants (au plus K) sur les nœuds suivants ; ce qui engendrera au total $O(KN^2)$ tâches d'ajout. Soit le coût total : $O(NK^2) + O(KN^2)$.

Listing 5.4 Algorithme du SPLIT

```
1
2  // nodes-vector est un vecteur de 0 à nb-tâches
3  SPLIT(nodes-vector, tâches-vector, nb-tâches)
4  {
5    top-node ←0;
6    last-node ←nb-tâches − 1;
7
8    while (top-node ≠nb-tâches)
9    {
10     /* Ne garder que nb-max-schedule-per-node sur le noeud nodes-vector[top-node] */
11     LIMIT-NODE(nodes-vector, top-node, nb-max-schedule-per-node);
12
13     /* Conserve uniquement la frontiere de Pareto sur le noeud nodes-vector[top-node] */
14     PARETO-FRONTIER(nodes-vector, top-node);
15
16     /* Sélectionner le meilleur ordonnancement avec ses intervalles de présence et la valeur du
           critère associée */
17     best-schedule, best-schedule-all-user-presence, best-schedule-criteria ←
           ARGMAX(nodes-vector[lastnode], CRITERIA);
18
19     /* Si le meilleur ordonnancement trouvé est de présence minimale alors c'est terminé */
20     if (best-schedule-all-user-presence = all-user-presence-min) return(best-schedule);
21
22     /* On garde dans le dernier noeud seulement le meilleur élement pour le critère */
23     nodes-vector[lastnode] ←LIST(best-schedule);
24
25     /* On distribue le noeud nodes-vector[top-node] sur les autres noeuds */
26     DISTRIBUTION(best-schedule-criteria, nodes-vector, top-node, tâches-vector, nb-tâches);
27
28     top-node ←top-node + 1;
29   }
30
31   /* Sélectionner le meilleur ordonnancement avec ses intervalles de présence et la valeur du critère
         associée */
32   best-schedule, best-schedule-all-user-presence, best-schedule-criteria ←
         ARGMAX(nodes-vector[lastnode], CRITERIA);
33
34   return(best-schedule);
35 }
```

5.8 Représentation des données

5.8.1 Les différentes tâches

Une *tâche simple* est une structure contenant la date de disponibilité, la durée et le nom de l'utilisateur de la tâche.

Listing 5.5 Distribution des séquences et des ordonnancements sur les nœuds

```
1  DISTRIBUTION(best-schedule-criteria, nodes-vector, top-node, tâches-vector, nb-tâches)
2  {
3    node ←nodes-vector[top-node];
4
5    // node est une liste d'ordonnancements
6    while (node ≠NIL)
7    {
8      current-schedule ←HEAD(node);
9      node ←TAIL(node);
10     DISTRIBUTION-PARTIAL-SCHEDULE(best-schedule-criteria, current-schedule, nodes-vector,
           top-node, tâches-vector, tâches-list);
11   }
12 }
13
14 DISTRIBUTION-PARTIAL-SCHEDULE(best-schedule-criteria, schedule, nodes-vector, top-node,
       tâches-vector, nb-tâches)
15 {
16   tâches-index ←top-node;
17   tâche ←tâches-vector[tâches-index];
18
19   /* On tient compte du théorème sur les tâches des utilisateurs permanents: il ne faut pas découper
         la séquence sur l'un d'eux */
20   if (UTILISATEUR-PERMANENT? tâche) return;
21
22   sequence ←START-SEQUENCE(tâche);
23   new-schedule ←SCHEDULE-ADD-SEQUENCE(schedule, sequence);
24
25   /* D'autre part on ne garde pas les ordonnancements infaisables (à cause du nombre de machines) */
26   if (SCHEDULE-INFEASIBLE? new-schedule) return;
27
28   tâches-index ←tâches-index + 1;
29
30   /* Pour finir on filtre les ordonnancements qui sont moins bons que le meilleur ordonnancement
         trouvé jusqu'ici dans le dernier noeud */
31   if (CRITERIA-BETTER new-schedule best-schedule-criteria)
         NODE-INSERT-SCHEDULE(nodes-vector, tâches-index+1, new-schedule);
32
33   while(tâches-index ≠nb-tâches)
34   {
35     // tâches-vector est un vecteur de 0 à nb-tâches-1
36     sequence ←EXTEND-SEQUENCE(sequence, tâches-vector[tâches-index]);
37     new-schedule ←SCHEDULE-ADD-SEQUENCE(schedule, sequence);
38
39     /* D'autre part on ne garde pas les ordonnancements infaisables (à cause du nombre de machines)
           */
40     if (SCHEDULE-INFEASIBLE? new-schedule) return;
41
42     tâches-index ←tâches-index + 1;
43
44     /* Pour finir on filtre les ordonnancements qui sont moins bons que le meilleur ordonnancement
           trouvé jusqu'ici dans le dernier noeud */
45     if (CRITERIA-BETTER new-schedule best-schedule-criteria)
           NODE-INSERT-SCHEDULE(nodes-vector, tâches-index, new-schedule);
46   }
47 }
```

Une *tâche placée* est une structure contenant en plus d'une simple tâche, une date de début d'exécution. On associe à chaque tâche placée un intervalle de présence de cette tâche $[disponibilite, execution + duree]$ dans le cas non périodique. Dans le cas périodique, il faut déterminer si la tâche provient de la période précédente ou non puis découper autour de la période pour obtenir une *liste* d'intervalles pour chaque morceaux de tâches.

Une *tâche générique* est une tâche périodique de période α quelconque. On calcule la période globale d'arrivée T des tâches en prenant le PPCM des périodes de toutes les tâches génériques.

Une *tâche de motif* est une tâche qu'on trouve une fois par motif. Sa structure contient sa date de disponibilité, sa durée, son utilisateur, son numéro de tâche générique et son index. Pour travailler dans un motif il faut répliquer les tâches génériques dans le motif $K = T/\alpha$ fois, tout en décalant les dates de disponibilité de α entre deux tâches de motif.

Une *tâche de motif placée* est une tâche qui compose un motif auquelle on associe une date de début d'exécution. Il faut déterminer si la tâche provient de la période précédente ou non en comparant sa date de disponibilité et sa date de début d'exécution. Ensuite, l'intervalle de présence de cette tâche peut-être obtenu en découpant autour de la période, on obtient alors une *liste* d'intervalles.

5.8.2 Les séquences

Si on considère un cycle du motif composé de n tâches, chacune des tâches peut soit être calée à sa date de disponibilité soit être placée à la suite d'une autre. On a donc pour un cycle du motif au pire 2^n possibilités de placement des tâches.

C'est pourquoi on considère plutôt les séquences du motif (ce qui peut aussi s'utiliser dans le cas non périodique) où la première tâche est calée à sa date de disponibilité et les autres à la suite.

On représente une séquence par sa date de début, sa date de fin ainsi que par la première tâche et la liste inversée des tâches suivantes qui la constitue. On a besoin de savoir de quoi est constitué une séquence pour ensuite calculer les intervalles de présence par utilisateur.

Dans le cas non périodique, l'ajout d'une tâche à une séquence ne peut se faire que si la tâche a une date de disponibilité inférieure à la fin de la séquence. Dans le cas périodique, l'ajout est toujours possible, la tâche pouvant provenir de la période précédente.

Dans le cas périodique, on part d'une séquence qu'il faut ensuite replier autour de la période pour obtenir des sequences *tronquées* (qui contiennent des tâches elles aussi *découpées*). Lorsqu'on découpe une séquence, les tâches peuvent être découpées sauf si une tâche se termine exactement à la fin d'une période.

De plus, contrairement au cas non périodique, on n'ajoute pas concrètement une séquence à une machine, mais on ajoute une liste de séquences tronquées à un ordonnancement car chaque morceau pourra être présent sur des machines distinctes. Peu importe le placement de ces morceaux sur les machines puisqu'on les considère comme des intervalles, avec la contrainte de respecter le nombre de machines. Il suffit alors de les placer sur la première machine possible.

5.8.3 Les machines

Une machine contient un ensemble de séquences disjointes. Dans le cas périodique, les séquences sont des séquences tronquées de tâches découpées.

Les opérations que l'on effectue sur une machine sont le test d'insertion d'une séquence, l'insertion d'une séquence et l'énumeration des séquences.

Nous avons choisi de représenter une machine comme un arbre binaire de recherche équilibré (un arbre AVL, avec insertion et test en $O(log(N))$, énumeration en $O(Nlog(N))$ pour N séquences dans la machine). L'ordre de comparaison naturel entre intervalle est utilisé.

Une représentation par un vecteur de taille T est possible, l'insertion, le test d'insertion et l'énumeration se font alors en $O(T)$, ce qui est intéressant lorsque les périodes sont petites.

5.8.3.a Les arbres de recherche binaires équilibrés (Arbres AVL)

Les arbres AVL [Adelson-Velskii et Landis, 1962], [Wirth, 1978], [Knuth, 1998] sont des arbres de recherches binaires équilibrés. Le nom AVL provient de leurs inventeurs : Georgy Adelson-Velskii et Evgenii Landis. L'idée consiste à maintenir un équilibre entre les hauteurs des sous-arbres d'un arbre binaire de recherche. Ainsi, chaque insertion ou suppression d'un élément doit assurer que l'arbre obtenu reste équilibré.

Nous utilisons des arbres AVL pour représenter des listes d'intervalles de présence contenant des opérations. La figure 5.8.1 illustre un arbre contenant 5 séquences. Cette structure de données permet une interrogation rapide en $O(log(N))$.

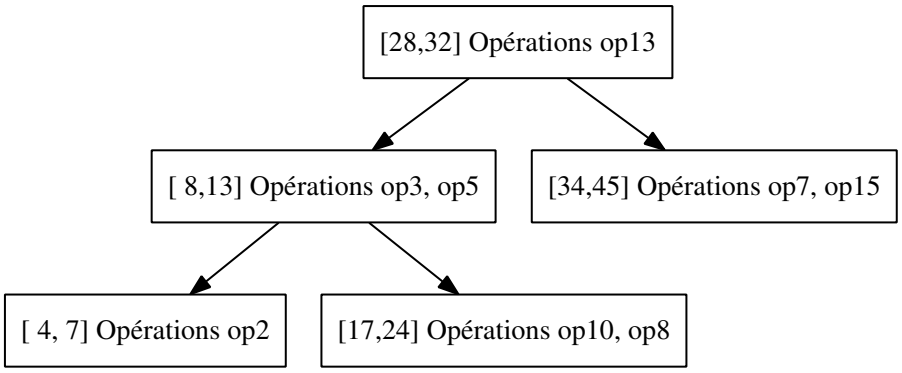


Fig. 5.8.1 Exemple de machine (Arbre AVL de séquences)

5.8.4 Un ordonnancement

La réalisation d'un ordonnancement est representée par un ensemble de m machines, celles-ci contiennent les séquences. Il peut être utile de travailler avec un nombre infini de machines, dans ce cas nous représentons un ordonnancement par une liste de machines.

Les séquences sont triées selon leur date de début. Lors de l'ajout d'une séquence à un ordonnancement, nous parcourons la liste des machines jusqu'à trouver une machine qui accepte la séquence.

Nous avons donc dans le cas non périodique la hiérarchie suivante :

Ordonnancement / Machine / Séquence / Tâche placée

Dans le cas périodique les séquences sont composées de tâches placées tronquées. Les tâches sont placées afin de connaître la date de disponibilité ainsi que la date de début d'exécution pour calculer les intervalles de présence. La hiérarchie devient :

Ordonnancement périodique / Machine périodique / Séquence tronquée / Tâche découpée / Tâche placée

Parmi les tâches sur un ordonnancement, il faut pouvoir énumérer les tâches placées dans l'ordonnancement, diviser un ordonnancement selon les utilisateurs et calculer les intervalles de présence de chaque utilisateur (ainsi que les intervalles de présence minimale).

Si un ordonnancement utilise au plus M machines, il peut se représenter sous la forme de M arbres de séquences disjointes (un arbre pour chaque machine). Inversement, soit une représentation sous la forme de M arbres de séquences, alors l'ordonnancement correspondant utilise moins de M machines.

5.9 Expérimentations numériques

Tâche	Utilisateur	Durée	Tâche	Utilisateur	Durée
1	1	10	14	2	50
2	1	70	15	2	13
3	1	38	16	2	84
4	1	85	17	2	62
5	1	3	18	2	30
6	1	87	19	2	91
7	1	53	20	2	22
8	1	43	21	2	97
9	1	27	22	2	2
10	1	64	23	2	49
11	1	15	24	2	24
12	1	72	25	2	83
13	1	73	26	2	60
27	3	55	36	4	97
28	3	85	37	4	9
29	3	7	38	4	99
30	3	37	39	4	35
31	3	21	40	4	83
32	3	26			
33	3	44			
34	3	12			
35	3	14			

Tab. 5.4 Problème considéré : 4 utilisateurs et 8 machines, période globale 3600. Toutes les tâches sont disponibles à la date 0.

Critère Max Limite	Temps (s)	Critère
10	2.1	(371, 674, 455, 589)
20	3.7	(244, 513, 448, 518)
30	6.3	(217, 338, 336, 306)
50	10.4	(270, 219, 300, 274)
100	21.5	(270, 254, 259, 274)
200	50.1	(244, 254, 259, 274)
300	69.9	(217, 250, 249, 246)
500	105.7	(217, 250, 249, 246)
1000	129.5	(217, 250, 249, 246)
Critère Total **Limite**	**Temps (s)**	**Critère**
10	2.1	2089
20	3.7	1723
30	6.2	1191
50	10.3	1063
100	21.3	1057
200	49.5	961
300	67.8	958
500	102.0	930
1000	125.2	930
Critère WeightedMax **Limite**	**Temps (s)**	**Critère**
10	2.1	2.239
20	3.7	1.614
30	6.2	1.123
50	10.4	0.677
100	21.4	0.677
200	49.7	0.671
300	70.3	0.628
500	110.6	0.628
1000	132.4	0.628
Critère WeightedTotal **Limite**	**Temps (s)**	**Critère**
10	2.1	4.99
20	3.7	3.89
30	6.2	2.77
50	10.3	2.38
100	21.4	2.38
200	50.2	2.14
300	70.4	2.12
500	104.4	1.93
1000	124.2	1.93

Tab. 5.5 Effet de la limitation du nombre de solutions partielles par nœud sur le temps de calcul et la qualité des solutions trouvées. Tour géant :

$$(30, 39, 34, 37, 24, 28, 7, 12, 32, 23, 2, 20, 5, 15, 33, 21, 13, 0, 4, 31,$$
$$16, 3, 35, 25, 6, 18, 1, 8, 36, 29, 11, 17, 38, 22, 14, 19, 10, 26, 9, 27)$$

Limite	Temps (s)	Critère
10000	326.6	944
500	92.8	1077
300	41.5	1389
100	15.5	1669
20	1.7	2180

Tab. 5.6 On ne garde que les K premiers sous-ordonnancements dans chaque nœud.

Limite	Temps (s)	Critère
10000	347.6	944
500	263.4	944
300	185.7	987
100	42.7	1012
20	3.5	1033

Tab. 5.7 On ne garde que les K derniers sous-ordonnancements dans chaque nœud. La solution est de meilleure qualité mais les temps sont plus longs. Pourtant, pour de petites valeurs de la limite et pour un temps donné, on trouve des solutions de qualité acceptable.

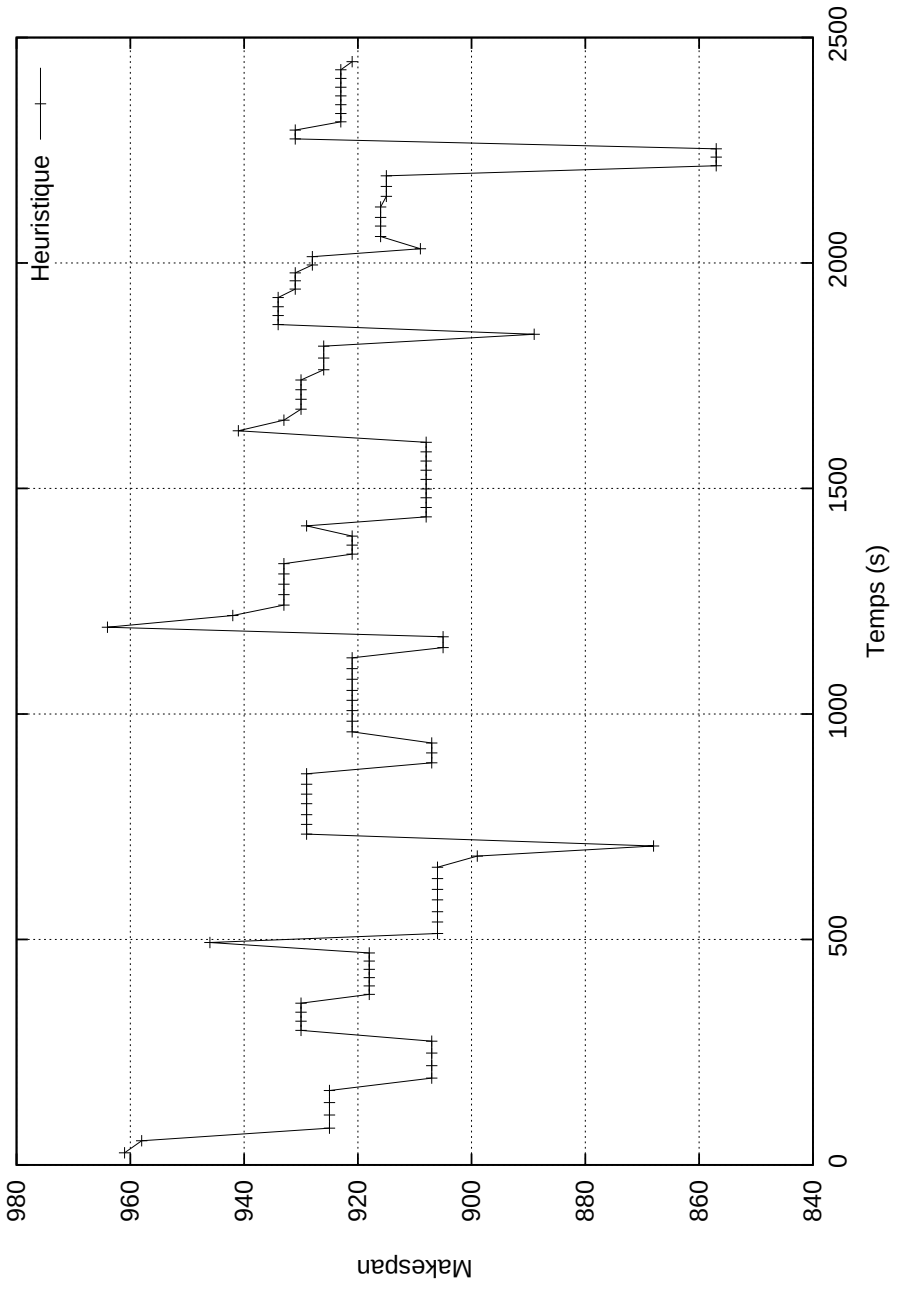


Fig. 5.9.1 Exemple d'exploration effectuée par l'heuristique sur un problème avec 4 utilisateurs, 8 machines et 40 opérations. Chaque utilisateur a ses tâches disponibles à la date 0. Le temps de présence maximal est équivalent au Makespan. On constate les successions de descente et de diversification de la solution. La meilleure solution trouvée est de Makespan égal à 857.

Critère Max Limite	Temps (s)	Critère
10	1.9	(761 726 508 693)
20	4.2	(312 277 378 244)
50	10.1	(312 277 378 244)
100	23.7	(269 218 277 259)
200	56.6	(258 218 240 259)
300	99.9	(258 218 240 259)
500	188.4	(258 218 240 259)
1000	333.6	(258 218 240 259)
Critère Total Limite	Temps (s)	Critère
10	1.9	2688
20	4.1	1197
50	10.1	1197
100	23.6	992
200	56.2	975
300	101.2	975
500	194.6	975
1000	335.6	975
Critère WeightedMax Limite	Temps (s)	Critère
10	1.9	2.41
20	4.9	0.88
50	10.2	0.88
100	24.2	0.79
200	58.8	0.60
300	103.5	0.60
500	190.9	0.60
1000	361.2	0.60
Critère WeightedTotal Limite	Temps (s)	Critère
10	1.9	6.61
20	4.2	2.74
50	10.1	2.74
100	23.7	2.33
200	57.4	2.23
300	104.1	2.12
500	198.9	2.12
1000	355.4	2.12

Tab. 5.8 Effet de la limitation du nombre de solutions partielles par nœud sur le temps de calcul et la qualité des solutions trouvées. Autre tour géant :

$(24, 26, 29, 39, 28, 2, 34, 7, 0, 13, 25, 4, 11, 18, 12, 35, 37, 14, 23, 15, 32, 31, 9, 19, 22, 5,$
$1, 6, 16, 36, 20, 17, 8, 21, 27, 3, 10, 33, 30, 38)$

Conclusion

> La paix est plus importante que toute la justice : la paix n'a pas été faite pour la justice, mais la justice pour la paix.
>
> Martin Luther King

L'objectif de ce travail de thèse a été de trouver un fondement scientifique à un principe qui refuse de négliger certains utilisateurs au bénéfice des autres : c'est le principe de l'équité.

Par conséquent, à partir de thèmes issus de la gestion des grilles, nous avons analysé plusieurs problèmes d'allocation et d'ordonnancement entre utilisateurs, sous l'angle de cette notion. L'originalité de notre travail a consisté, entre autres, à prendre en considération chaque utilisateur et à mesurer pour chacun la performance obtenue. Cette approche nous a conduit à considérer, non un critère scalaire, mais un critère vectoriel avec une composante par utilisateur.

Après avoir exploré différents points de vue sur la notion d'équité, nous nous sommes basés sur un ensemble d'axiomes qui nous ont semblé souhaitables ; ceci nous a conduit à l'ordre Leximin et à en étudier ses propriétés.

Nous nous sommes ensuite intéressés aux problèmes d'allocation de ressources à différents utilisateurs, égaux en droits, lorsque les demandes sont incertaines et évoluent dans le temps. Nous avons défini les ensembles de politiques POC et POE et fourni une caractérisation de l'ensemble des POC. Nous avons montré des propriétés de l'ensemble des POE ; ce qui nous a permis de proposer un algorithme d'énumération de ces politiques. Cet algorithme peut être utilisé en ligne et donne une politique POE sur le scénario effectif.

Par la suite, nous avons considéré les problèmes d'ordonnancement de tâches périodiques multi-utilisateurs. Un critère indépendant au regroupement de tâches a été proposé ; lequel se base sur les intervalles de présence des utilisateurs. Un premier modèle linéaire a été élaboré ; celui-ci calcule la phase transitoire jusqu'à l'apparition d'un motif. Puis, un second modèle, qui ne calcule que le motif périodique, a été élaboré. Ce dernier, de taille plus modeste, nous a permis de résoudre des problèmes comportant jusqu'à une vingtaine de tâches.

Afin de proposer des solutions de qualité raisonnable lorsque le problème est de plus grande taille, nous avons conçu une heuristique basée sur l'idée de l'algorithme de calcul SPLIT pour les tournées de véhicules. Plusieurs règles de dominances nous ont permis de réduire l'espace de recherche.

Pour résumer : ce qui importe est de choisir judicieusement ses objectifs et ses priorités. Les notions d'impartialité ou d'équité peuvent paraître secondaires, voire impopulaires, pourtant elles fournissent des garanties de service et des gages de confiance. Voulons-nous seulement améliorer tels ou tels aspects d'un problème ? Le critère d'équité n'est-il pas à prendre en compte lorsque le problème comporte plusieurs utilisateurs ? Voici des questions que chacun devrait être amené à se poser.

Questions ouvertes et perspectives

Allocation de ressources

Une des future directions possible à ces recherches consisterait à prendre en compte des ressources de durée d'utilisation finie, telles que des tâches dans une grille de calcul. Une extension du modèle d'allocation pourrait être de prendre en compte l'évolution du nombre de machines au cours du temps ou bien la possibilité pour les utilisateurs de céder une partie de leur allocation entre deux points de décisions lorsqu'ils ne l'utilisent plus. L'étude d'allocation multi-ressources serait une autre piste d'étude intéressante. En effet, il est possible de devoir partager plusieurs types de ressources aux participants ceux-ci ayant des besoins variés selon la ressource. Ces besoins en ressources pourraient être corrélés, comme par exemple des besoins en CPU, en mémoire, en stockage ou en bande passante pour des tâches sur une grille de calcul. Certaines tâches ne pourront pas s'éxécuter par manque d'espace de stockage ou de mémoire, de même il peut être impossible d'accepter plus de tâches si la bande passante en entrée ou en sortie est insuffisante, etc. Notons que les ressources elles-mêmes peuvent être hétérogènes, ce qui complique leur répartition. Enfin, il serait possible d'envisager le cas de politiques d'allocation aléatoires, mais dans ce cas se pose alors la question de l'évaluation de ces politiques.

Ordonnancement périodique

Une question reste ouverte : les ordonnancements à exécution périodique sont-ils dominants pour le critère donné ? Tous les ordonnancements dont les tâches arrivent périodiquement, ne sont pas obligatoirement périodiques. Il serait intéressant d'étudier pour quelles fonctions d'objectif, l'ordonnancement optimal est toujours périodique lorsque les périodes sont rationnelles.

On pourrait généraliser le problème en autorisant la préemption des tâches, lorsque cela s'y prête. Comment alors modifier les programmes linéaires ?

Une autre extension d'étude serait de considérer, en plus des tâches périodiques, des tâches dites *sporadiques* du point de vue de l'équité. Ces tâches sporadiques peuvent s'intercaler sans modifier les tâches périodiques, soit demander à perturber l'exécution périodique afin de les exécuter.

Dans une prochaine étape, nous désirons appliquer les résultats obtenus, et proposer des outils en les adaptant dans le cadre de la grille de calcul.

Bibliographie

Adelson-Velskii, G. et Landis, E. : 1962, An algorithm for the organization of information, *Doklady Akademii Nauk USSR* **146**(2), 263–266.

Agnetis, A., de Pascale, G. et Pacciarelli, D. : 2009, A lagrangian approach to single-machine scheduling problems with two competing agents, *J. Scheduling* **12**(4), 401–415.

Agnetis, A., Pacciarelli, D. et Pacifici, A. : 2007, Multi-agent single machine scheduling, *Annals OR* **150**(1), 3–15.

Agnetis, A., Pacciarelli, D. et Pacifici, A. : 2010, Combinatorial models for multi-agent scheduling problems, *12th int. conf. on Project Management and Scheduling, Tours (France)* pp. 37–40.

Ahmed, M. H., Anpalagan, A., Chen, K.-C., Han, Z. et Hossain, E. : 2009, Fairness in radio resource management for wireless networks, *EURASIP Journal on Wireless Communications and Networking* **2009**.

Ajtai, M., Aspnes, J., Naor, M., Rabani, Y., Schulman, L. J. et Waarts, O. : 1995, Fairness in scheduling, *SODA '95 : Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 477–485.

Allais, M. : 1953, Le comportement de l'homme rationnel devant le risque : Critique des postulats et axiomes de l'école americaine, *Econometrica* **21**(4), 503–546.

Allman, M., Paxson, V. et Stevens, W. : 1999, TCP Congestion Control, *RFC 2581* .

Andreetto, P., Andreozzi, S., Avellino, G., Beco, S., Cavallini, A., Cecchi, M., Ciaschini, V., Dorise, A., Giacomini, F., Gianelle, A., Grandinetti, U., Guarise, A., Krop, A., Lops, R., Maraschini, A., Martelli, V., Marzolla, M., Mezzadri, M., Molinari, E., Monforte, S., Pacini, F., Pappalardo, M., Parrini, A., Patania, G., Petronzio, L., Piro, R., Porciani, M., Prelz, F., Rebatto, D., Ronchieri, E., Sgaravatto, M., Venturi, V. et Zangrando, L. : 2008, The glite workload management system, *Journal of Physics : Conference Series* **119**(6), 062007.

Angel, E., Bampis, E., Pascual, F. et Tchetgnia, A.-A. : 2009, On truthfulness and approximation for scheduling selfish tasks, *Journal of Scheduling* **12**(5), 437–445.

Aoun, D., Déplanche, A.-M. et Trinquet, Y. : 2008, Pfair scheduling improvement to reduce interprocessor migrations, *in* Giorgio Buttazzo et Pascale Minet (eds), *16th International Conference on Real-Time and Network Systems (RTNS 2008)*, Isabelle Puaut, Rennes France. C. : Computer Systems Organization/C.3 : SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS/C.3.2 : Real-time and embedded systems, J. : Computer Applications/J.7 : COMPUTERS IN OTHER SYSTEMS/J.7.6 : Real time.

Aristote : -340, *Éthique à Nicomaque*.

Arnold, B. C. : 1987, *Majorization and the Lorenz Order : A Brief Introduction*, Vol. 43 of *Lecture Notes in Statistics*, Springer-Verlag, Berlin.

Arrow, K. J. : 1970, *Social Choice and Individual Values, Second edition (Cowles Foundation Monographs Series)*, 2nd edn, Yale University Press.

Artigues, C., Leus, R. et Herroelen, W. : 2005, Allocation de resources en planification de projects pour la construction d'ordonnancements robustes, *in* J. C. Billaut, A. Moukrim et E. Sanlaville (eds), *Flexibilité et robustesse en ordonnancement*, Traité IC2 Information-Commande-Communication, Informatique et Systèmes d'Informations, Hermès Science, pp. 163 – 189. ISBN 2-7462-1028-2.

Artigues, C., Leus, R. et Herroelen, W. : 2008, Resource allocation for the construction of robust project schedules, *in* J.-C. Billaut, A. Moukrim et E. Sanlaville (eds), *Flexibility and robustness in scheduling*, Control systems, robotics and manufacturing series, ISTE-WILEY, pp. 171–197. ISBN 978-1-84821-054-7.

Avi-Itzhak, B., Levy, H. et Raz, D. : 2007, A resource allocation queueing fairness measure : Properties and bounds, *Queueing Systems Theory and Application* **56**(2), 65–71.

Avrachenkov, K., Elias, J., Martignon, F., Neglia, G. et Petrosyan, L. : 2010, A Nash bargaining solution for Cooperative Network Formation Games, *Technical Report RR-7480*, INRIA.

Axelrod, R. M. : 1984, *The evolution of cooperation*, Basic Books, New York.

Ayesta, U., Brun, O. et Prabhu, B. : 2009, Price of Anarchy in Non-Cooperative Load Balancing. 09833.

Azar, Y. : 1992, On-line load balancing, *Theoretical Computer Science*, Springer, pp. 218–225.

Azar, Y., Broder, A. Z., Karlin, A. R. et Upfal, E. : 2000, Balanced allocations, *SIAM J. Comput.* **29**(1), 180–200.

Azar, Y. et Epstein, L. : 1997, On-line machine covering, *Proceedings of the 5th Annual European Symposium on Algorithms*, Springer-Verlag, London, UK, pp. 23–36.

Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K. R. et Waarts, O. : 1997, On-line load balancing of temporary tasks, *Journal of Algorithms* **22**(1), 93 – 110.

Baker, K. R. et Smith, J. C. : 2003, A multiple-criterion model for machine scheduling, *Journal of Scheduling* **6**(1), 7–16.

Balasubramanian, H., Fowler, J. W., Keha, A. B. et Pfund, M. E. : 2009, Scheduling interfering job sets on parallel machines, *European Journal of Operational Research* **199**(1), 55–67.

Bansal, N., Dhamdhere, K., Könemann, J. et Sinha, A. : 2004, Non-clairvoyant scheduling for minimizing mean slowdown, *Algorithmica* **40**(4), 305–318.

Bar-Noy, A., Mayer, A., Schieber, B. et Sudan, M. : 1998, Guaranteeing fair service to persistent dependent tasks, *SIAM J. Comput.* **27**(4), 1168–1189.

Baruah, S. K., Cohen, N. K., Plaxton, C. G. et Varvel, D. A. : 1993, Proportionate progress : a notion of fairness in resource allocation, *STOC '93 : Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, ACM, New York, NY, USA, pp. 345–354.

Bauchet, P. : 1965, Pareto (vilfredo) - cours d'économie politique, *Revue Économique* **16**(5), 811–812.

Bawa, V. S. : 1975, Optimal rules for ordering uncertain prospects, *Journal of Financial Economics* **2**(1), 95–121.

Beasley, J. : 1983, Route first–cluster second methods for vehicle routing, *Omega* **11**(4), 403–408.

Beaufils, B. et Mathieu, P. : 2002, Faut-il toujours suivre Nash ?, *in* P. Mathieu et J.-P. Müller (eds), *Systèmes multi-agents et systèmes complexes (JFIADSMA'2002)*, Hermès, pp. 17–31.

Bender, M. A., Chakrabarti, S. et Muthukrishnan, S. : 1998, Flow and stretch metrics for scheduling continuous job streams, *SODA '98 : Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 270–279.

Bender, M. A., Muthukrishnan, S. et Rajaraman, R. : 2002, Improved algorithms for stretch scheduling, *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 762–771.

Bender, M., Muthukrishnan, S. et Rajaraman, R. : 2004, Approximation algorithms for average stretch scheduling, *Journal of Scheduling* **7**, 195–222. 10.1023/B :JOSH.0000019681.52701.8b.

Bentham, J. : 1823, *An introduction to the principles of morals and legislation*, new ed. / corr. by the author. edn, Printed for W. Pickering and R. Wilson, London :.

Berger, E. D., Zorn, B. G. et McKinley, K. S. : 2002, Reconsidering custom memory allocation, *OOPSLA '02 : Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM, New York, NY, USA, pp. 1–12.

Berten, V. et Gaujal, B. : 2005, Index routing for task allocation in grids, *Technical Report RR-5892*, INRIA.

Berten, V. et Gaujal, B. : 2007, Brokering strategies in computational grids using stochastic prediction models, *Parallel Comput.* **33**(4-5), 238–249.

Berten, V., Goossens, J. et Jeannot, E. : 2006, On the distribution of sequential jobs in random brokering for heterogeneous computational grids, *IEEE Transactions on Parallel and Distributed Systems* **15**(2), 113–124.

Bertsekas, D. et Gallager, R. : 1987, *Data networks*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bhargava, R., Goel, A. et Meyerson, A. : 2001, Using approximate majorization to characterize protocol fairness, *SIGMETRICS Perform. Eval. Rev.* **29**, 330–331.

Billaut, J.-C., Moukrim, A. et Sanlaville, E. : 2008, *Flexibility and Robustness in Scheduling*, ISTE Ltd and John Wiley & Sons, Inc.

Billaut, J.-C., Moukrim, A. et Sanlaville, E. (eds) : 2005, *Flexibilité et la robustesse en ordonnancement*, Hermès, chapter Introduction à la flexibilité et la robustesse en ordonnancement, pp. 15–34.

Binmore, K. : 1991, *Fun and Games - A Text on Game Theory*, D. C. Heath & Co.

Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G. et Vahi, K. : 2003, The role of planning in grid computing, *in* E. Giunchiglia, N. Muscettola et D. S. Nau (eds), *ICAPS*, AAAI, pp. 153–163.

Boudec, J.-Y. : 2000, Rate adaptation, congestion control and fairness : A tutorial.

Bouveret, S. et Lemaître, M. : 2006, Un algorithme de programmation par contraintes pour la recherche d'allocations leximin-optimales, *in* AFPC (ed.), *Actes des deuxièmes Journées Francophones de Programmation par Contraintes (JFPC'06)*, Nîmes, France. (in french).

Brakerski, Z., Nisgav, A. et Patt-Shamir, B. : 2002, General perfectly periodic scheduling, *PODC '02 : Proceedings of the twenty-first annual symposium on Principles of distributed computing*, ACM, New York, NY, USA, pp. 163–172.

Brucker, P. : 2001, *Scheduling Algorithms*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bruno, J., Coffman, Jr., E. G. et Sethi, R. : 1974, Scheduling independent tasks to reduce mean finishing time, *Commun. ACM* **17**(7), 382–387.

Capit, N., Costa, G. D., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P. et Richard, O. : 2005, A batch scheduler with high level components, *CCGRID*, IEEE Computer Society, pp. 776–783.

Casanova, H. : 2006, On the harmfulness of redundant batch requests, *High-Performance Distributed Computing, International Symposium on* **0**, 255–266.

Casanova, H., Desprez, F. et Suter, F. : 2010, On Cluster Resource Allocation for Multiple Parallel Task Graphs, *Journal of Parallel and Distributed Computing* **70**(12), 1193–1203.

Cavalieri, S., Monforte, S. et Scibilia, F. : 2003, Proposing and evaluating allocation algorithms in a grid environment, *in* P. Sloot, D. Abramson, A. Bogdanov, Y. Gorbachev, J. Dongarra et A. Zomaya (eds), *Computational Science - ICCS 2003*, Vol. 2659 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 701–701.

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget, J., Phelps, S., Rodriguez-Aguilar, J. A. et Sousa, P. : 2006, Issues in multiagent resource allocation.

Chevaleyre, Y., Endriss, U., Lang, J. et Maudet, N. : 2007, A short introduction to computational social choice, *SOFSEM '07 : 33rd conference on Current Trends in Theory and Practice of Computer Science*, Springer, pp. 51–69.

Chiu, D. M. : 1999, Some observations on fairness of bandwidth sharing, *Technical report*, Department of Information Engineering , The Chinese University of Hong Kong, Mountain View, CA, USA.

Christodoulopoulos, K., Gkamas, V. et Varvarigos, E. A. : 2008, Statistical analysis and modeling of jobs in a grid environment, *J. Grid Comput.* **6**(1), 77–101.

Cravo, G. L., Ribeiro, G. M. et Lorena, L. A. N. : 2008, A greedy randomized adaptive search procedure for the point-feature cartographic label placement, *Comput. Geosci.* **34**(4), 373–386.

Csirik, J., Kellerer, H. et Woeginger, G. : 1992, The exact lpt-bound for maximizing the minimum completion time, *Operations Research Letters* **11**(5), 281 – 287.

Czumaj, A., Krysta, P. et Vöcking, B. : 2002, Selfish traffic allocation for server farms, *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, ACM, New York, NY, USA, pp. 287–296.

D. M. Ryan, C. H. et Glover, F. : 1993, Extensions of the petal method for vehicle routing, *Journal of the Operational Research Society* **44**, 289–296.

D'Alembert, J. : 1761, Sur le calcul des probabilités., *Opuscules Mathématiques* **4**.

D'Alembert, J. : 1764, Réflexions sur le calcul des probabilités., *Opuscules Mathématiques* **2**(10).

Dalton, H. : 1920, The Measurement of the Inequality of Incomes, *The Economic Journal* **30**(119), 348–361.

Dantzig, G. B. et Ramser, J. H. : 1959, The truck dispatching problem, *Management Science* **6**(1), 80–91.

Debreu, G. : 1954, Representation of a preference ordering by a numerical function, *in* R. Thrall, C. Coombs et R. Davies (eds), *Decision Processes*, Wiley, New York, pp. 159–175.

Denda, R., Banchs, A. et Effelsberg, W. : 2000, The fairness challenge in computer networks, *Proceedings of the First COST 263 International Workshop on Quality of Future Internet Services*, QofIS '00, Springer-Verlag, London, UK, pp. 208–220.

Deng, J., Han, Y. S. et Liang, B. : 2009, Fairness index based on variational distance, *Proceedings of the 28th IEEE conference on Global telecommunications*, GLOBECOM'09, IEEE Press, Piscataway, NJ, USA, pp. 3338–3343.

Deschamps, R. et Gevers, L. : 1978, Leximin and utilitarian rules : A joint characterization, *Journal of Economic Theory* **17**(2), 143 – 163.

Deschinkel, K. et Touati, S.-A.-A. : 2008, Efficient method for periodic task scheduling with storage requirement minimization, *Proceedings of the 2nd international conference on Combinatorial Optimization and Applications*, COCOA 2008, Springer-Verlag, Berlin, Heidelberg, pp. 438–447.

Dhakal, S., Hayat, M. M., Pezoa, J. E., Yang, C. et Bader, D. A. : 2007, Dynamic load balancing in distributed systems in the presence of delays : A regeneration-theory approach, *IEEE Trans. Parallel Distrib. Syst.* **18**, 485–497.

Dinechin, B. D. D. : 1994, Simplex scheduling : More than lifetime-sensitive instruction scheduling, *Technical report*, Proceedings of the International Conference on Parallel Architecture and Compiler Techniques.

Du, J. et Leung, J. Y. : 1990, Minimizing total tardiness on one machine is np-hard, *Math. Oper. Res.* **15**(3), 483–495.

Dubois, D. et Prade, H. : 1996, Semantics of quotient operators in fuzzy relational databases, *Fuzzy Sets and Systems* **78**(1), 89 – 93.

Duhamel, C., Lacomme, P., Prins, C. et Prodhon, C. : 2010, A grasp-els approach for the capacitated location-routing problem, *Computers & OR* **37**(11), 1912–1923.

Duhamel, C., Lacomme, P. et Prodhon, C. : 2011, Efficient frameworks for greedy split and new depth first search split procedures for routing problems, *Computers & Operations Research* **38**, 723–739.

Eager, D. L., Lazowska, E. D. et Zahorjan, J. : 1986, Adaptive load sharing in homogeneous distributed systems, *IEEE Trans. Softw. Eng.* **12**, 662–675.

Ehrgott, M. : 1995, Lexicographic max-ordering - a solution concept for multicriteria combinatorial optimization, *in* D. Schweigert (ed.), *Methods of Multicriteria Decision Theory, Proceedings of the 5th Workshop of the DGOR-Working Group Multicriteria Optimization and Decision Theory*, pp. 55–66.

Ehrgott, M. : 1996, A characterization of lexicographic max-ordering solutions, *Technical report*, University of Kaiserslautern, Department of Mathematics.

Ehrgott, M. : 2000, *Multicriteria optimization*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.

Elkind, E. et Lang, J. : 2011, Guest editorial : special issue on computational social choice, *Autonomous Agents and Multi-Agent Systems* **22**, 1–3. 10.1007/s10458-010-9155-0.

Ellsberg, D. : 1961, Risk, Ambiguity, and the Savage Axioms, *The Quarterly Journal of Economics* **75**(4), 643–669.

Epstein, L. et Van Stee, R. : 2008, Maximizing the minimum load for selfish agents, *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, Springer-Verlag, Berlin, Heidelberg, pp. 264–275.

Erdil, D. et Lewis, M. : 2010, Dynamic grid load sharing with adaptive dissemination protocols, *The Journal of Supercomputing* pp. 1–28. 10.1007/s11227-010-0507-y.

Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R. et Streit, A. : 2002, On advantages of grid computing for parallel job scheduling, *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, IEEE Computer Society, Washington, DC, USA, pp. 39–.

Esswein, C., Billaut, J.-C. et Artigues, C. : 2008, Scheduling groups of operations : a multicriteria approach to provide sequential flexibility, *in* J.-C. Billaut, A. Moukrim et E. Sanlaville (eds), *Flexibility and robustness in scheduling*, Control systems, robotics and manufacturing series, ISTE-WILEY, pp. 227–248. ISBN 978-1-84821-054-7.

Fabozzi, F. J., Kolm, P. N., Pachamanova, D. et Focardi, S. M. : 2007, *Robust Portfolio Optimization and Management*, Wiley, Hoboken, NJ.

Fagin, R. et Williams, J. H. : 1983, A fair carpool scheduling algorithm, *IBM J. Res. Dev.* **27**(2), 133–139.

Fargier, H., Lang, J., Lemaître, M. et Verfaillie, G. : 2004, Partage équitable de ressources communes. (1) Un modèle général et son application au partage de ressources satellitaires, *Technique et Science Informatiques* **23**(9), 1187–1217.

Feitelson, D. G., Rudolph, L. et Schwiegelshohn, U. : 2004, Parallel job scheduling - a status report, *in* D. G. Feitelson, L. Rudolph et U. Schwiegelshohn (eds), *JSSPP*, Vol. 3277 of *Lecture Notes in Computer Science*, Springer, pp. 1–16.

Feitelson, D. et Rudolph, L. : 1995, Parallel job scheduling : Issues and approaches, *in* D. Feitelson et L. Rudolph (eds), *Job Scheduling Strategies for Parallel Processing*, Vol. 949 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–18.

Feng, H., Misra, V. et Rubenstein, D. : 2007, Pbs : a unified priority-based scheduler, *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, ACM, New York, NY, USA, pp. 203–214.

Feo, T. A. et Resende, M. G. : 1995, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**, 109–133.

Feo, T. A., Venkatraman, K. et Bard, J. F. : 1991, A GRASP(tm) for a difficult single machine scheduling problem,, *Computers & Operations Research* **18**(8), 635 – 643.

Feo, T. et Bard, J. : 1989, Flight scheduling and maintenance base planning, *Management Science* **35**, 1415–1432.

Festa, P. et Resende, M. : 2002, GRASP : An annotated bibliography, *in* C. Ribeiro et P. Hansen (eds), *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, pp. 325–367.

Fimmel, D. et Müller, J. : 2002, A flow graph formulation of optimal software pipelining.

Fimmel, D. et Müller, J. : 2001, Optimal software pipelining under resource constraints, *Int. J. Found. Comput. Sci.* **12**(6), 697–718.

Fishburn, P. : 1982, *The Foundations of Expected Utility*, D. Reidel Publishing, Dordrecht.

Fishburn, P. C. : 1964, "*Decision and Value Theory*", Publications in operations research, New York, Wiley.

Fleurbaey, M. : 2008, *Fairness, responsibility, and welfare / Marc Fleurbaey*, Oxford ; New York : Oxford University Press. Formerly CIP.

Floyd, S. : 2000, Congestion Control Principles, IETF Request For Comment 2914 (Best Current Practice).

Francez, N. : 1986, *Fairness*, Springer-Verlag New York, Inc., New York, NY, USA.

Frankfurt, H. : 1987, Equality as a moral ideal, *Ethics* **98**(1), 21–43.

Funke, B., Grünert, T. et Irnich, S. : 2005, Local search for vehicle routing and scheduling problems : Review and conceptual integration, *Journal of Heuristics* **11**(4), 267–306.

Garey, M. R. et Johnson, D. S. : 1975, Complexity results for multiprocessor scheduling under resource constraints, *SIAM Journal on Computing* **4**(4), 397–411.

Garey, M. R. et Johnson, D. S. : 1990, *Computers and Intractability ; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA.

Gentzsch, W. : 2001, Sun grid engine : Towards creating a compute power grid, *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, IEEE Computer Society, Washington, DC, USA, pp. 35–36.

Glover, F. W. et Kochenberger, G. A. : 2003, *Handbook of Metaheuristics*, Vol. 57 of *International series in operations research and management science*, Kluwer Academic Publishers, Boston Hardbound.

Goel, A. et Meyerson, A. : 2006, Simultaneous optimization via approximate majorization for concave profits or convex costs, *Algorithmica* **44**(4), 301–323.

Goel, A., Meyerson, A. et Plotkin, S. : 2001a, Approximate majorization and fair online load balancing, *SODA '01 : Proceedings of the twelfth annual ACM-SIAM Symposium On Discrete Algorithms*, pp. 384–390.

Goel, A., Meyerson, A. et Plotkin, S. : 2001b, Combining fairness with throughput : Online routing with multiple objectives., *J. Comput. Syst. Sci.* **63**(1), 62–79.

Golovin, D. : 2005, Max-min fair allocation of indivisible goods, *Technical Report CMU-CS-05-144*, School of Computer Science, Carnegie Mellon University.

Gottinger, H. W. : 1982, Foundations of lexicographic utility, *Mathematical Social Sciences* **3**(4), 363 – 371.

Govindarajan, R., Altman, E. R. et Gao, G. R. : 1996, A framework for resource-constrained rate-optimal software pipelining, *IEEE Trans. Parallel Distrib. Syst.* **7**, 1133–1149.

Graham, R., Lawler, E., Lenstra, J. et Kan, A. : 1979, Optimization and approximation in deterministic sequencing and scheduling : a survey, *in* E. J. P.L. Hammer et B. Korte (eds), *Discrete Optimization II, Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, Vol. 5 of *Annals of Discrete Mathematics*, Elsevier, pp. 287 – 326.

Gromicho, J., van, J. H., Kok, A. et Schutten, J. : 2009, Restricted dynamic programming : a flexible framework for solving realistic vrps.

Guilbaud, G. T. : 1952, Les théories de l'intérêt général et le problème logique de l'agrégation, *Économie appliquée* pp. 501–584.

Guitart, J., Torres, J. et Ayguadé, E. : 2010, A survey on performance management for internet applications, *Concurr. Comput. : Pract. Exper.* **22**, 68–106.

Hadar, J. et Russell, W. R. : 1969, Rules for ordering uncertain prospects, *American Economic Review* **59**(1), 25–34.

Hahne, E. L. : 1991, Round-robin scheduling for max-min fairness in data networks, *IEEE Journal on Selected Areas in Communications* **9**, 1024–1039.

Hanen, C. et Munier, A. : 1995, A study of the cyclic scheduling problem on parallel processors, *Discrete Appl. Math.* **57**, 167–192.

Haouari, M. et Jemmali, M. : 2008, Maximizing the minimum completion time on parallel machines, *4OR : A Quarterly Journal of Operations Research* **6**, 375–392. 10.1007/s10288-007-0053-5.

Hardin, G. : 1968, The tragedy of the commons, *Science* **162**, 1243–1248.

Hardy, G. H., Littlewood, J. E. et Pólya, G. : 1934, *Inequalities*, The University press, Cambridge [Eng.].

Hardy, G. H., Littlewood, J. E. et Polya, G. : 1929, Some simple inequalities satisfied by convex functions, *Messenger Math.* **58**, 145–152.

Hardy, G. H. et Ramanujan, S. : 1918, Asymptotic formulae in combinatory analysis., *Proc. London Math. Soc.* **17**(2), 75–115.

Hart, J. et Shogan, A. : 1987, Semi-greedy heuristics : An empirical study, *Operations Research Letters* **6**, 107–114.

Haskins, R. : 2001, ISPAdmin, *;LOGIN : The USENIX Magazine* **26**(1).

Henry, C. : 2005, Du risque à l'incertitude dans les modèles de décisions. CECO-292.

Hites, R., De Smet, Y., Risse, N., Salazar-Neumann, M. et Vincke, P. : 2006, About the applicability of mcda to some robustness problems., *European Journal of Operational Research* **174**(1), 322–332.

Horn, W. : 1973, Minimizing average flow time with parallel machines., *Oper. Res.* **21**, 846–847.

Huynh Tuong, N., Soukhal, A. et Billaut, J.-C. : 2009, New scheduling problems with interfering and independent jobs. 33 pages. Paper submitted to Journal of scheduling the 8 September 2009.

Ito, K., Lucke, L. E. et Parhi, K. K. : 1998, Ilp-based cost-optimal dsp synthesis with module selection and data format conversion, *IEEE Trans. Very Large Scale Integr. Syst.* **6**, 582–594.

Jackson, D., Snell, Q. et Clement, M. : 2001, Core algorithms of the maui scheduler, *in* D. Feitelson et L. Rudolph (eds), *Job Scheduling Strategies for Parallel Processing*, Vol. 2221 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 87–102. 10.1007/3-540-45540-X_6.

Jackson, J. R. : 1955, Scheduling a production line to minimize maximum tardiness, *Research Report 53*, Management Science, University of California, Los Angeles, CA.

Jacq, N., Breton, V., Chen, H.-Y., Ho, L.-Y., Hofmann, M., Lee, H.-C., Legré, Y., Lin, S.-C., Maass, A., Medernach, E., Merelli, I., Milanesi, L., Rastelli, G., Reichstadt, M., Salzemann, J., Schwichtenberg, H., Sridhar, M., Kasam, V., Wu, Y.-T. et Zimmermann, M. : 2006, Large Scale

In Silico Screening on Grid Infrastructures, *The Third International Life Science Grid Workshop, LSGrid 2006*, Yokohama Japon. 14 pages, 2 figures, 2 tables, The Third International Life Science Grid Workshop, LSGrid 2006, Yokohama, Japan, 13-14 october 2006, to appear in the proceedings - PCSV, présenté par N. Jacq à paraître dans les proceedings.

Jacq, N., Breton, V., Chen, H.-Y., Ho, L.-Y., Hofmann, M., Lee, H.-C., Legré, Y., Lin, S., Maaß, A., Medernach, E., Merelli, I., Milanesi, L., Rastelli, G., Reichstadt, M., Salzemann, J., Schwichtenberg, H., Sridhar, M., Kasam, V., Wu, Y.-T. et Zimmermann, M. : 2007, Grid-Enabled High Throughput Virtual Screening, pp. 45–59.

Jaffe, J. M. : 1981, Bottleneck flow control, *IEEE Transactions on Communications* **29**(7), 954–962.

Jagabathula, S. et Shah, D. : 2008, Fair scheduling in networks through packet election, *CoRR* **abs/0808.2530**.

Jain, R., Chiu, D. et Hawe, W. : 1984, A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, *Technical Report TR-301*, DEC Research.

Jeffay, K., Stanat, D. F. et Martel, C. U. : 1991, On Non-Preemptive Scheduling of Periodic and Sporadic Tasks, *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pp. 129–139.

Jones, J. P. : 2002, Beowulf cluster computing with linux, MIT Press, Cambridge, MA, USA, chapter PBS : portable batch system, pp. 369–390.

Kalai, E. et Smorodinsky, M. : 1975, Other solutions to nash's bargaining problem, *Econometrica* **43**(3), 513–18.

Kameda, H. et Altman, E. : 2008, Inefficient noncooperation in networking games of common-pool resources, *IEEE Journal on Selected Areas in Communications* **26**(7), 1260–1268.

Kannan, S., Roberts, M., Mayes, P., Brelsford, D. et Skovira, J. F. : 2001, *Workload Management with LoadLeveler*, IBM Press.

Kay, J. et Lauder, P. : 1988, A fair share scheduler, *Commun. ACM* **31**(1), 44–55.

Kelly, F. P. : 1997, Charging and rate control for elastic traffic, *European Transactions on Telecommunications* **8**, 33–37.

Kelly, F. P., Maulloo, A. K. et Tan, D. K. H. : 1998, Rate control for communication networks : shadow prices, proportional fairness and stability, *Journal of the Operational Research Society* **49**(3), 237–252.

Keynes, J. M. : 1921, *A treatise on probability*, Macmillan, London,.

Keynes, J. M. : 1926, *The end of laissez-faire*, Published by L. & V. Woolf at The Hogarth Press, London :.

Keynes, J. M. : 1936, *The General theory of employment, interest and money*, Macmillan, London :.

Knight, F. H. : n.d., *Risk, Uncertainty and Profit*.

Knuth, D. E. : 1998, *The art of computer programming, volume 3 : (2nd ed.) sorting and searching*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Kolm, S. : 1997, *Justice and equity*, MIT Press, Cambridge, Mass. [u.a.].

Kostreva, M. M. et Ogryczak, W. : 1999, Linear optimization with multiple equitable criteria, *RAIRO Operations Research* **33**, 275–297.

Kostreva, M. M., Ogryczak, W. et Wierzbicki, A. : 2004, Equitable aggregations and multiple criteria analysis, *European Journal of Operational Research* **158**(2), 362–377. Methodological Foundations of Multi-Criteria Decision Making.

Kouvelis, P. et Yu, G. : 1997, *Robust discrete optimization and its applications*, Kluwer Academic Publishers, Dordrecht.

Kremien, O., Kramer, J. et Magee, J. : 1993, Scalable, adaptive load sharing for distributed systems, *IEEE Parallel Distrib. Technol.* **1**, 62–70.

Kum, K.-I. et Sung, W. : 2001, Combined word-length optimization and high-level synthesis of digital signal processing systems, *Computer-Aided Design of Intergrated Circuits and Systems* **20**, 921–930.

Kumar, A. et Kleinberg, J. : 2000, Fairness measures for resource allocation, *Foundations of Computer Science, Annual IEEE Symposium on* **0**, 75.

Kyselovà, D., Dubois, D., Komorníková, M. et Mesiar, R. : 2007, Refining aggregation operator-based orderings in multifactorial evaluation - part i : Continuous scales, *IEEE T. Fuzzy Systems* **15**(6), 1100–1106.

L., B., Brandenburger, A. et Dekel, E. : 1989, An overview of lexicographic choice under uncertainty, *Ann. Oper. Res.* **19**, 231–246.

Laguna, M. et Velarde, J. L. G. : 1991, A search heuristic for just-in-time scheduling in parallel machines, *Journal of Intelligent Manufacturing* **2**, 253–260. 10.1007/BF01471113.

Lahrichi, A. : 1982, Scheduling methods. the notion of 'compulsory parts' and its application to cumulative problems., *Comptes Rendus de l'Academie des Sciences* **294**, 209–211.

Lai, T.-C. et Kuo, Y.-K. : 1996, Minimizing total tardiness for single machine sequencing, *Journal of the Operations Research Society of Japan* **39**(3).

Lam, T.-W., Ting, H.-F., To, K.-K. et Wong, W.-H. : 2002, On-line load balancing of temporary tasks revisited, *Theor. Comput. Sci.* **270**, 325–340.

Lan, T., Gao, D., Chiang, M. et Sabharwal, A. : 2010, An axiomatic theory of fairness, *IEEE INFOCOM* .

Landesman, B. M. : 1983, Egalitarianism, *Canadian Journal of Philosophy* **13**(1), 27 – 56.

Larmouth, J. : 1974, Scheduling for a share of the machine, *Technical Report UCAM-CL-TR-2*, University of Cambridge, Computer Laboratory.

Larmouth, J. : 1978, Scheduling for immediate turnaround, *Software – Practice and Experience* **8**, 559–578.

Larson, R. C. : 1987, Perspective on queues : Social justice and the psychology of queueing, *Operations Research* **35**, 895–905.

Larson, R. C. : 1998, Beyond the physics of queueing, *CORS/INFORMS Montréal, Quebec*, Massachusetts Institute of Technology (MIT).

Lauder, P. : 1995, The share scheduler revisited, *Technical report*.

Laure, E., Fisher, S. M., Frohner, A., Grandi, C., Kunszt, P. Z., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Di Meglio, A. et Edlund, A. : 2006, Programming the grid with glite, *Technical Report EGEE-TR-2006-001*, CERN, Geneva.

Laure, E., Hemmer, F., Prelz, F., Beco, S., Fisher, S., Livny, M., Guy, L., Barroso, M., Buncic, P., Kunszt, P. Z., Di Meglio, A., Aimar, A., Edlund, A., Groep, D., Pacini, F., Sgaravatto, M. et Mulmo, O. : 2004, Middleware for the next generation grid infrastructure, (EGEE-PUB-2004-002), 4 p.

Lawler, E. L. : 1982, Recent results in the theory of machine scheduling., *Mathematical programming : the state of the art* pp. 202–234.

Lawler, E. L. et Martel, C. U. : 1981, Scheduling periodically occurring tasks on multiple processors, *Inf. Process. Lett.* **12**(1), 9–12.

Lazarev, A., Sadykov, R. et Sevastyanov, S. : 2007, A scheme of approximation solution of problem $1|r_i|l_{\max}$, *Journal of Applied and Industrial Mathematics* **1**, 468–480. 10.1134/S1990478907040102.

Legrand, A., Su, A. et Vivien, F. : 2005, Minimizing the stretch when scheduling flows of biological requests, *Research Report RR-5724*, INRIA.

Legrand, A., Su, A. et Vivien, F. : 2008, Minimizing the stretch when scheduling flows of divisible requests, *Journal of Scheduling* .

Lenstra, J. K., Rinnooy Kan, A. H. G. et Brucker, P. : 1977, Complexity of machine scheduling problems, *Ann. of Discrete Math.* **1**, 343–362.

Li, H. et Wolters, L. : 2007, Towards a better understanding of workload dynamics on data-intensive clusters and grids, *Parallel and Distributed Processing Symposium, International* **0**, 60.

Lin, G.-H., Yao, E.-Y. et He, Y. : 1998, Parallel machine scheduling to maximize the minimum load with nonsimultaneous machine available times, *Operations Research Letters* **22**(2-3), 75 – 81.

Lingrand, D., Montagnat, J. et Glatard, T. : 2009, Deriving grid workload models from user submission strategies. Research Report I3S laboratory, number I3S/RR-2009-17-FR, Sophia Antipolis.

Lipton, R. J., Markakis, E., Mossel, E. et Saberi, A. : 2004, On approximately fair allocations of indivisible goods, *EC '04 : Proceedings of the 5th ACM conference on Electronic commerce*, ACM, New York, NY, USA, pp. 125–131.

Liu, C. L. et Layland, J. W. : 1973, Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM* **20**(1), 46–61.

Macías, M., Smith, G., Rana, O., Guitart, J. et Torres, J. : 2010, Enforcing service level agreements using an economically enhanced resource manager, *in* D. Neumann, M. Baker, J. Altmann et O. Rana (eds), *Economic Models and Algorithms for Distributed Systems*, Autonomic Systems, Birkhäuser Basel, pp. 109–127.

Maeno, T. : 2008, Panda : distributed production and distributed analysis system for atlas, *Journal of Physics : Conference Series* **119**(6), 062036.

Maniquet, F. : 2002, Social orderings for the assignment of indivisible objects, *Economics Working Papers 0015*, Institute for Advanced Study, School of Social Science, Princeton.

Mann, L. : 1969, Queue culture : The waiting line as a social system, *Am. J. Sociol.* **75**, 340–354.

Marsh, M. T. et Schilling, D. A. : 1994, Equity measurement in facility location analysis : A review and framework, *European Journal of Operational Research* **74**(1), 1–17.

Massoulie, L. et Roberts, J. : 1999, Bandwidth sharing : Objectives and algorithms, *IEEE/ACM Transactions on Networking*, pp. 1395–1403.

Massoulié, L. et Roberts, J. : 2000, Bandwidth sharing and admission control for elastic traffic, *Telecommunication Systems* **15**(1-2), 185–201.

McNaughton, R. : 1959, Scheduling with deadlines and loss functions, *Management Science* **6**(1), 1–12.

Medernach, E. : 2005, Workload Analysis of a Cluster in a Grid Environment, *in* D. Feitelson, E. Frachtenberg, L. Rudolph et U. Schwiegelshohn (eds), *Job Scheduling Strategies for Parallel Processing*, Vol. 3834 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Berlin, Heidelberg, chapter 2, pp. 36–61.

Megiddo, N. : 1977, A good algorithm for lexicographically optimal flows in multi-terminal networks, *Bulletin of the American Mathematical Society* **83**(3), 407–409.

Meiners, C. R. et Torng, E. : 2007, Mixed criteria packet scheduling, *Proceedings of the 3rd international conference on Algorithmic Aspects in Information and Management*, AAIM '07, Springer-Verlag, Berlin, Heidelberg, pp. 120–133.

Mill, J. S. : 1863, *Utilitarianism*, History of Economic Thought Books, McMaster University Archive for the History of Economic Thought.

Mor, B. et Mosheiov, G. : 2010, Scheduling problems with two competing agents to minimize minmax and minsum earliness measures, *European Journal of Operational Research* **206**(3), 540 – 546.

Mortimore, G. W. : 1968, An ideal of equality, *Mind* **77**(306), 222–242.

Moscicki, J. T., Brochu, F., Ebke, J., Egede, U., Elmsheuser, J., Harrison, K., Jones, R. W. L., Lee, H. C., Liko, S., Maier, A., Muraru, A., Patrick, G. N., Pajchel, K., Reece, W., Samset, B. H., Slater, M. W., Soroko, A., Tan, C. L., van der Ster, D. C. et Williams, M. : 2009, Ganga : a tool for computational-task management and easy access to grid resources, *Comput. Phys. Commun.* **180**(arXiv :0902.2685. 11), 2303–2316.

Motwani, R., Phillips, S. et Torng, E. : 1993, Non-clairvoyant scheduling, *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 422–431.

Moulin, H. : 1991, *Axioms of Cooperative Decision Making*, Econometric Society Monographs, Cambridge University Press.

Munier, A. : 1996, The basic cyclic scheduling problem with linear precedence constraints, *Discrete Appl. Math.* **64**, 219–238.

Myerson, R. : 1991, *Game theory : analysis of conflict*, Vol. 3, Harvard University Press, Cambridge, MA, Boston.

Nace, D., Pioro, M. et Doan, L.-N. : 2006, A tutorial on max-min fairness and its applications to routing, load-balancing and network design, *4th IEEE International Conference on Computer Sciences Research, Innovation and Vision for the Future (RIVF 2006)*.

Naqvi, S., Mouton, S., Massonet, P., Silaghi, G. C., Battré, D., Hovestadt, M. et Djemame, K. : 2008, Using sla based approach to handle sabotage tolerance in the grids, *in* T. Priol et M. Vanneschi (eds), *From Grids to Service and Pervasive Computing*, Springer US, pp. 153–162.

Nash, J. F. : 1950a, The bargaining problem, *Econometrica* **18**(2), 155–162.

Nash, J. F. : 1950b, Equilibrium points in n-person games, *Proceedings of the National Academy of Sciences of the United States of America* **36**(1), 48–49.

Nash, J. F. : 1951, Non-cooperative games, *Annals of Mathematics* **54**(2), 286–295.

Neumann, J. V. et Morgenstern, O. : 1944, *Theory of Games and Economic Behavior*, Princeton University Press.

Newbury, J. P. : 1982, Immediate turnround - an elusive goal, *Software : Practice and Experience* **12**, 897–906.

Nieh, J., Vaill, C. et Zhong, H. : 2001, Virtual-time round-robin : An o(1) proportional share scheduler, *in* Y. Park (ed.), *USENIX Annual Technical Conference, General Track*, USENIX, pp. 245–259.

Nielsen, N. R. : 1970, The allocation of computer resources—is pricing the answer ?, *Commun. ACM* **13**(8), 467–474.

Norman, D. A. : 2008, The psychology of waiting lines.

Ogryczak, W. : 1998, Location problems from the multiple criteria perspective : Efficient solutions, *Archives of Control Sciences* **7**, 161–180.

Ogryczak, W., Wierzbicki, A. et Milewski, M. : 2008, A multi-criteria approach to fair and efficient bandwidth allocation, *Omega* **36**(3), 451–463.

Ostrom, E. : 2008, Tragedy of the commons, *The New Palgrave Dictionary of Economics*, Palgrave Macmillan, New York.

Pacini, F. : 2005, JDL Attributes Specification, *Technical report*.

Papadimitriou, C. et Valiant, G. : 2009, Revisiting the price of anarchy in selfish routing.

Park, D. : 1980, On the semantics of fair parallelism, *in* D. Bjørner (ed.), *Abstract Software Specifications*, Vol. 86 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 504–526.

Parsons, E. W. et Sevcik, K. C. : 1997, Implementing multiprocessor scheduling disciplines, *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS '97, Springer-Verlag, London, UK, pp. 166–192.

Pascual, F., Rzadca, K. et Trystram, D. : 2009, Cooperation in multi-organization scheduling, *Concurr. Comput. : Pract. Exper.* **21**(7), 905–921.

Paterson, S. K. et Tsaregorodtsev, A. : 2008, Dirac optimized workload management, *Journal of Physics : Conference Series* **119**(6), 062040.

Pigou, A. C. : 1912, *Wealth and welfare*, Macmillan and co., limited.

Poder, E., Beldiceanu, N. et Sanlaville, E. : 2004, Computing a lower approximation of the compulsory part of a task with varying duration and varying resource consumption, *European Journal of Operational Research* **153**, 239–254. ISSN 0377-2217.

Prodan, R. et Wieczorek, M. : 2010, Negotiation-based scheduling of scientific grid workflows through advance reservations, *Journal of Grid Computing* **8**, 493–510. 10.1007/s10723-010-9165-9.

Pruhs, K., Sgall, J. et Torng, E. : 2004, Online scheduling, *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*, CRC Press, Inc., Boca Raton, FL, USA, chapter 15.

Rafaeli, A., Barron, G. et Haber, K. : 2002, The effects of queue structure on attitudes, *Journal of Service Research* **5**(2), 125–139.

Ramezani, S. et Endriss, U. : 2010, Nash social welfare in multiagent resource allocation, *Agent-Mediated Electronic Commerce : Designing Trading Strategies and Mechanisms for Electronic Markets*, Vol. 59 of *Lecture Notes in Business Information Processing*, Springer-Verlag, pp. 117–131. Postproceedings of AMEC-2009. Also presented at BNAIC-2009.

Rawls, J. : 1971, *A Theory of Justice*, Vol. 1, Harvard University Press.

Rawls, J. : 1985, Justice as fairness : Political not metaphysical, *Philosophy and Public Affairs* **14**(3), 223–251.

Rayward-Smith, V. J., Osman, I. H., Reeves, C. R. et Smith, G. D. (eds) : 1996, *Modern Heuristic Search Methods*, Wiley.

Raz, D., Avi-Itzhak, B. et Levy, H. : 2006, Fairness considerations of scheduling in multi-server and multi-queue systems, *valuetools '06 : Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, ACM, New York, NY, USA, p. 39.

Raz, D., Levy, H. et Avi-Itzhak, B. : 2004, RAQFM : A resource allocation queueing fairness measure, *Technical Report RRR-32-2004*, RUTCOR, Rutgers University.

Reeves, C. : 1995, *Modern Heuristic Techniques for Combinatorial Problems*, Advances topics in computer science, Mc Graw-Hill.

Resende, M. : 2000, Greedy randomized adaptive search procedures (GRASP), *Technical Report TR 98.41.1*, AT&T Labs-Research.

Rockafellar, R. T. et Wets, R. J.-B. : 1991, Scenarios and policy aggregation in optimization under uncertainty, *Math. Oper. Res.* **16**, 119–147.

Roy, B. : 2007, Double pondération pour calculer une moyenne : Pourquoi et comment ?, *RAIRO Operations Research* **41**(2).

Roy, B. et Bouyssou, D. : 1993, *Aide Multicritère à la Décision : Méthodes et Cas*, Economica, Paris.

Rzadca, K. : 2007, Scheduling in multi-organization grids : Measuring the inefficiency of decentralization, *PPAM 2007 Proceedings*, number 4967 in *LNCS*, Springer, pp. 1048–1058.

Rzadca, K., Trystram, D. et Wierzbicki, A. : 2007, Fair game-theoretic resource management in dedicated grids, *IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), Proceedings*, IEEE Computer Society.

Sabin, G., Sahasrabudhe, V. et Sadayappan, P. : 2004, On fairness in distributed job scheduling across multiple sites., *CLUSTER*, IEEE Computer Society, pp. 35–44.

Sadykov, R. et Lazarev, A. : 2005, Experimental comparison of branch-and-bound algorithms for the $1|r_i|l_{\max}$ problem, *Book of Abstracts, 7th International Workshop on Models and Algorithms for Planning and Scheduling Problems* pp. 239–241.

Salles, R. M. et Barria, J. A. : 2008, Lexicographic maximin optimisation for fair bandwidth allocation in computer networks, *European Journal of Operational Research* **185**, 778–794.

Samuelson, P. A. : 1977, St. petersburg paradoxes : Defanged, dissected, and historically described, *Journal of Economic Literature* **15**(1), 24–55.

Sandel, M. J. : 1984, The procedural republic and the unencumbered self, *Political Theory* **12**(1), 81–96.

Sandholm, T., Lai, K. et Clearwater, S. : 2008, Admission control in a computational market, *Cluster Computing and the Grid, IEEE International Symposium on* **0**, 277–286.

Sandmann, W. : 2006, Scheduling to improve queue justice, *Proceedings of the 20th European Conference on Modelling and Simulation, ECMS 2006*, Bonn, Germany, pp. 372–377.

Saule, E. et Trystram, D. : 2009, Multiusers scheduling in parallel systems, *int. symposium on parallel and distributed processing*, IEEE, Roma, pp. 1–9.

Sauve, J., Wong, J. et Field, J. : 1980, On fairness in packet-switching networks, *Proceedings of the 21st IEEE Computer Society International Conference* pp. 466–470.

Schur, I. : 1923, Uber eine klasse von mittlebildungen mit anwendungen auf der determinantentheorie, *Sitzungsber. Berlin, Math. Gesellshaft* **22**, 9–29.

Schutten, J. : 1996, List scheduling revisited, *Operations Research Letters* **18**(4), 167–170.

Schwiegelshohn, U. et Yahyapour, R. : 2000, Fairness in parallel job scheduling, *Journal of Scheduling* **3**(5), 297–320.

Sen, A. : 1970, *Collective Choice and Social Welfare*, North Holland, Amsterdam.

Shmoys, D. B., Wein, J. et Williamson, D. P. : 1995, Scheduling parallel machines on-line, *SIAM J. Comput.* **24**(6), 1313–1331.

Skutella, M. et Verschae, J. : 2009, A robust ptas for the parallel machine covering problem, *in* C. Barnhart, U. Clausen, U. Lauther et R. H. Möhring (eds), *Models and Algorithms for Optimization in Logistics*, number 09261 in *Dagstuhl Seminar Proceedings*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany.

Staples, G. : 2006, Torque resource manager, *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, ACM, New York, NY, USA.

Stillwell, M., Schanzenbach, D., Vivien, F. et Casanova, H. : 2009, Resource allocation using virtual clusters, *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, IEEE Computer Society, Washington, DC, USA, pp. 260–267.

Stillwell, M., Vivien, F. et Casanova, H. : 2010, Dynamic fractional resource scheduling for hpc workloads, *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE CS Press.

Taflin, E. : 1999, Equity allocation and portfolio selection in insurance, *Quantitative Finance Papers math/9907160*, arXiv.org.

Tan, Z. et Gurd, J. R. : 2007, Market-based grid resource allocation using a stable continuous double auction, *Grid Computing, 2007 8th IEEE/ACM International Conference on* pp. 283–290.

Tan, Z. et He, Y. : 2004, Ordinal scheduling problem and its asymptotically optimal algorithms on parallel machine system, *Science in China Series F : Information Sciences* **47**, 161–169. 10.1360/01yf0571.

Tannenbaum, T., Wright, D., Miller, K. et Livny, M. : 2001, Condor – a distributed job scheduler, *in* T. Sterling (ed.), *Beowulf Cluster Computing with Linux*, MIT Press.

Tarhan, B. et Grossmann, I. E. : 2008, A multistage stochastic programming approach with strategies for uncertainty reduction in the synthesis of process networks with uncertain yields, *Computers & Chemical Engineering* **32**(4-5), 766–788.

Taylor, A. D. : 2004, Hervé moulin, fair division and collective welfare, *Public Choice* **119**(3_-4), 468–470.

Thain, D., Tannenbaum, T. et Livny, M. : 2005, Distributed computing in practice : the condor experience., *Concurrency - Practice and Experience* **17**(2-4), 323–356.

Thomson, W. : 2007, Lorenz rankings of rules for the adjudication of conflicting claims, *RCER Working Papers 538*, University of Rochester - Center for Economic Research (RCER).

Touati, C., Altman, E. et Galtier, J. : 2001, On fairness in bandwidth allocation, *Technical Report RR-4269*, INRIA.

Touati, C., Altman, E. et Galtier, J. : 2006, Generalized Nash bargaining solution for bandwidth allocation, *Computer Networks* **50**(17), 3242–3263.

Tsaregorodtsev, A., Garonne, V. et Stokes-Rees, I. : 2004, Dirac : A scalable lightweight architecture for high throughput computing, *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, IEEE Computer Society, Washington, DC, USA, pp. 19–25.

Van Dongen, V., Gao, G. et Ning, Q. : 1992, A polynomial time method for optimal software pipelining, *in* L. Bougé, M. Cosnard, Y. Robert et D. Trystram (eds), *Parallel Processing : CONPAR 92-VAPP V*, Vol. 634 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 613–624.

Vanderster, D. C., Elmsheuser, J., Liko, D., Maeno, T., Nilsson, P., Wenaus, T. et Walker, R. : 2010, A panda backend for the ganga analysis interface, *Journal of Physics : Conference Series* **219**(7), 072056.

Varacca, D. et Völzer, H. : 2006, New perspectives on fairness, *Bulletin of the European Association for Theoretical Computer Science* **90**, 90–108.

Wang, L., Fang, L. et Hipel, K. W. : 2004, Lexicographic minimax approach to fair water allocation problems, *SMC (1)*, IEEE, pp. 1038–1043.

Welzl, M., Papadimitriou, D., Scharf, M. et Briscoe, B. : 2010, Open research issues in internet congestion control, *Internet Draft draft-irtf-iccrg-welzl-congestion-control-open-research-06*, IETF. (Work in progress).

Whitmore, G. : 1989, "stochastic dominance for the class of completely monotonic utility functions", *Studies in the Economics of Uncertainty : In Honor of Josef Hadar, Thomas B. Fomby and Tae Kun Seo* pp. 77 – 88.

Wierman, A. : 2007, Fairness and classifications, *SIGMETRICS Perform. Eval. Rev.* **34**, 4–12.

Wilson, P., Johnstone, M., Neely, M. et Boles, D. : 1995, Dynamic storage allocation : A survey and critical review, *in* H. Baler (ed.), *Memory Management*, Vol. 986 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–116. 10.1007/3-540-60368-9_19.

Wirth, N. : 1978, *Algorithms + Data Structures = Programs*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Woeginger, G. J. : 1997, A polynomial-time approximation scheme for maximizing the minimum machine completion time, *Operations Research Letters* **20**(4), 149 – 154.

Wong, C. S., Tan, I., Kumari, R. D. et Wey, F. : 2008, Towards achieving fairness in the linux scheduler, *SIGOPS Oper. Syst. Rev.* **42**(5), 34–43.

Wong, J. et Lang, S. : 1982, Queueing network models of packet switching networks, part 1 : Open networks, *Performance Evaluation* .

Wong, J., Sauve, F. et Field, J. : 1982, A study of fairness in packet-switching networks, *IEEE Transaction on Communications* **30**(2), 346–353.

Woodside, C. M. : 1986, Controllability of computer performance tradeoffs obtained using controlled-share queue schedulers, *IEEE Trans. Softw. Eng.* **12**(10), 1041–1048.

Yager, R. R. : 1988, On ordered weighted averaging aggregation operators in multicriteria decisionmaking, *IEEE Trans. Syst. Man Cybern.* **18**, 183–190.

Yager, R. R. : 1997, On the analytic representation of the leximin ordering and its application to flexible constraint propagation, *European Journal of Operational Research* **102**(1), 176 – 192.

Yoo, A., Jette, M. et Grondona, M. : 2003, Slurm : Simple linux utility for resource management, *in* D. Feitelson, L. Rudolph et U. Schwiegelshohn (eds), *Job Scheduling Strategies for Parallel Processing*, Vol. 2862 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 44–60.

Young, H. P. : 1994, *Equity in Theory and Practice*, Russell Sage Foundation, Princeton University Press.

Zhao, H. et Sakellariou, R. : 2006, Scheduling multiple dags onto heterogeneous systems, *Parallel and Distributed Processing Symposium, IEEE* p. 130.